SEARCH-BASED PREDICTION OF SOFTWARE QUALITY:

EVALUATIONS AND COMPARISONS

Wasif Afzal

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2011:06

School of Computing

Search-Based Prediction of Software Quality:

Evaluations And Comparisons

Wasif Afzal

Blekinge Institute of Technology Doctoral Dissertation Series
No 2011:06
ISSN 1653-2090
ISBN 978-91-7295-203-4

Search-Based Prediction of Software Quality:

Evaluations And Comparisons

Wasif Afzal

School of Computing
Blekinge Institute of Technology
SWEDEN

School of Computing
Publisher: Blekinge Institute of Technology
Printed by Printfabriken, Karlskrona, Sweden 2011
ISBN 978-91-7295-203-4
Blekinge Institute of Technology Doctoral Dissertation Series
ISSN 1653-2090
urn:nbn:se:bth-00490

To my father

There is no such thing as a failed experiment,
only experiments with unexpected outcomes.
Richard Buckminster Fuller (1895–1983)

ABSTRACT

Software verification and validation (V&V) activities are critical for achieving software quality; however, these activities also constitute a large part of the costs when developing software. Therefore efficient and effective software V&V activities are both a priority and a necessity considering the pressure to decrease time-to-market and the intense competition faced by many, if not all, companies today. It is then perhaps not unexpected that decisions that affects software quality, e.g., how to allocate testing resources, develop testing schedules and to decide when to stop testing, needs to be as stable and accurate as possible.

The objective of this thesis is to investigate how search-based techniques can support decision-making and help control variation in software V&V activities, thereby indirectly improving software quality. Several themes in providing this support are investigated: predicting reliability of future software versions based on fault history; fault prediction to improve test phase efficiency; assignment of resources to fixing faults; and distinguishing fault-prone software modules from non-faulty ones. A common element in these investigations is the use of search-based techniques, often also called metaheuristic techniques, for supporting the V&V decision-making processes. Search-based techniques are promising since, as many problems in real world, software V&V can be formulated as optimization problems where near optimal solutions are often good enough. Moreover, these techniques are general optimization solutions that can potentially be applied across a larger variety of decision-making situations than other existing alternatives. Apart from presenting the current state of the art, in the form of a systematic literature review, and doing comparative evaluations of a variety of metaheuristic techniques on large-scale projects (both industrial and open-source), this thesis also presents methodological investigations using search-based techniques that are relevant to the task of software quality measurement and prediction.

The results of applying search-based techniques in large-scale projects, while investigating a variety of research themes, show that they consistently give competitive results in comparison with existing techniques. Based on the research findings, we conclude that search-based techniques are viable techniques to use in supporting the decision-making processes within software V&V activities. The accuracy and consistency of these techniques make them important tools when developing future decision-support for effective management of software V&V activities.

ACKNOWLEDGMENTS

This thesis would not have happened without the invaluable support, inspiration and guidance of my advisors, Dr. Richard Torkar and Dr. Robert Feldt. I am lucky to have them supervising me; their ideas have been instrumental in steering this research. Thank you R & R, I have still to learn a lot from you two! I am also thankful to Prof. Claes Wohlin for allowing me the opportunity to undertake post-graduate studies and to be part of the SERL research group.

Thank you Prof. Lee Altenberg at The University of Hawai'i at Mānoa for offering the course that made me learn about the fascinating field of evolutionary computation. My gratitude also goes to Dr. Junchao Xiao at the Institute of Software at The Chinese Academy of Sciences in Beijing, China, for collaborating on a research paper and having me as a guest in China. I am also thankful to Prof. Anneliese Amschler Andrews at The University of Denver and Dr. Greger Wikstrand at Know IT Yahm AB for taking interest in my research and providing useful comments. Many thanks to Prof. Mark Harman for inviting me over to his research group (CREST) for a month.

I am also thankful to our industrial contacts for being responsive to requests for data sets. Especially, it would not have been possible to complete this thesis without the help of ST-Ericsson in Lund and Sauer-Danfoss in Älmhult. Many thanks to Conor White and Evert Nilsson from ST-Ericsson and Fredrik Björn from Sauer-Danfoss, for taking time out to support my research.

I also appreciate the organizers of the annual symposium on search-based software engineering for providing a fantastic platform to present research results and to have fruitful discussions for future work.

I also appreciate the feedback of anonymous reviewers on earlier drafts of publications in this thesis.

My colleagues at the SERL research group have been supportive throughout the research. Dr. Tony Gorschek has been very kind to read a few of my papers and in providing feedback. I am thankful to my fellow Ph.D. students (both part of SERL and the Swedish research school in verification and validation, SWELL) for offering me learning opportunities through interaction and course work, especially Michael Unterkalmsteiner, for encouraging me to co-author yet another systematic mapping study.

It will be unjust not to mention the support I got from the library staff, especially Kent Pettersson and Eva Norling helped me find research papers and books on several occasions. Additionally, Kent Adolfsson, Camilla Eriksson, May-Louise Andersson, Eleonore Lundberg, Monica H. Nilsson and Anna P. Nilsson provided administrative support whenever it was required.

I will remain indebted to my family for providing me the confidence and comfort to undertake post-graduate studies overseas. I thank my mother, brother and sisters for their support and backing. I would like to thank my nephews and nieces for coloring my life. I will remain grateful to my father who passed away in 2003 but not before he had influenced my personality to be what I am today. Lastly, I thank my wife, Fizza, for being an immediate source of joy and love, for providing me the late impetus to complete this thesis on a happier note.

The research presented in this thesis was funded partly by Sparbanksstiftelsen Kronan, NordForsk, The Knowledge Foundation and the Swedish research school in verification and validation (SWELL) through Vinnova (the Swedish Governmental Agency for Innovation Systems).

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 Preamble

The IEEE Standard Glossary of Software Engineering Terminology [301] defines software engineering as: “(1) The application of a systematic, disciplined, quantifiable approach to the *development*, *operation*, and *maintenance* of software; that is, the application of engineering to software. (2) The study of approaches as in (1)”. Within software *development* different phases constitutes a software development life cycle, with the objective of translating end user needs into a software product. Typical phases include concept, requirements definition, design, implementation and test. During the course of a software development life cycle, certain surrounding activities [273] occur, and software verification and validation (V&V) is the name given to one set of such activities. The collection of software V&V activities is also often termed as software quality assurance (SQA) activities.

Software verification consists of activities that check the correct implementation of a specific function, while software validation consists of activities that check if the software satisfies customer requirements. The IEEE Guide for Software Verification and Validation Plans [299] precisely illustrates this as: “A V&V effort strives to ensure that quality is built into the software and that the software satisfies user requirements.” Boehm [34] presented another way to state the distinction between software V&V:

Verification: “Are we building the product right?”
Validation: “Are we building the right product?”

Example software V&V activities include formal technical reviews, inspections, walk-throughs, audits, testing and techniques for software quality measurement.

Another possible way to understand software V&V activities is to categorize them into static and dynamic techniques complemented with different ways to conduct software quality measurements. Static techniques examine software artifacts without executing them (examples include inspections and reviews) while dynamic techniques (software testing) executes the software to identify quality issues. Software quality measurement approaches, on the other hand, helps in the management decision-making process (examples include assistance in deciding when to stop testing [141]).

The overarching purpose of software V&V activities is to improve software product quality. At the heart of a high-quality software product is variation control [273]:

> *From one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment and calendar time. In general, we would like to make sure our testing program covers a known percentage of the software, from one release to another. Not only do we want to minimize the number of defects that are released to the field, we'd like to ensure that the variance in the number of bugs is also minimized from one release to another.*

It is then reasonable to argue that controlling variation in software V&V activities increases our chances of delivering quality software to end-users. Efficient and cost-effective management of software V&V activities is one of the challenging tasks of software project management and considerable gains can be made when considering that software V&V activities constitute a fair percentage of the total software development life cycle costs; according to Boehm and Basili, around 40% [36], while Myers [247] argues that detection and removal of faults constitutes around 50% of project budgets.

There are other reasons to motivate a better management of V&V activities. We live today in a competitive global economy where time-to-market is of utmost importance [275]. At the same time, the size and complexity of software developed today, is constantly increasing. Releasing a software product then has to be a trade-off between the time-to-market, cost-effectiveness and the quality levels built into the software. We believe that efficient and cost-effective software V&V activities can help management make such a trade-off. The aim of this thesis is to help management decision-making processes in controlling variation during software V&V activities, thereby supporting software quality. We expect management to, by part through our studies, gain support in decision-making, regarding an assessment of the quality level of the software under test. This can in turn be used for assessment of testing schedule slippage, decisions related to testing resource allocation and reaching an agreement on when to stop testing and preparing for shipping the software. Our argument for such a decision-support is based on measures that support software quality. To achieve this, we investigate multiple related themes in this thesis:

- The possibility of analyzing software fault[1] history as a measurement technique to predict future software reliability.
- Using measures to support test phase efficiency.
- Using measures to support assignment of resources to fix faults.
- Using measures to classify fault-prone parts of the software from non fault-prone parts.

Apart from the above themes being within the domain of software V&V, one other common element, in the investigation of the above themes, is the use of search-based techniques. These techniques represent computational methods that iteratively try to optimize a candidate solution with respect to a certain measure of quality and represent an active field of research within the broad domain of artificial intelligence (AI) [57]. Examples of such techniques include simulated annealing, genetic algorithms, genetic programming, ant colony optimization and artificial immune systems. The focus on search-based techniques also grows out of an increasing interest in an emerging field within software engineering called search-based software engineering (SBSE) [128, 131]. SBSE seeks to reformulate software engineering problems as search-based problems, thereby facilitating the application of search-based techniques. Investigating such techniques is useful since they represent general optimization solutions that can potentially be applied across a larger variety of decision-making situations. This thesis, in essence, is an evaluation of the use of such techniques for supporting the decision-making processes within software V&V activities. Figure 1.1 presents an overview of the major concerns addressed in this thesis.

This thesis consists of either published or submitted research papers that have been edited for the purpose of forming chapters in this thesis. This editing includes removing repetitions only, except for Chapter 3, which is an aggregation of three research papers and involve changes in the structure and discussion of results. The introductory chapter is organized as follows. Section 1.2 introduces the concepts and related work for this thesis. Section 1.3 describes the application of search-based software engineering in our work. Section 1.4 presents the research questions that were posed during the work on this thesis. This section also covers the main contributions of the thesis. Section 1.5 presents the research methodology as used in this thesis. Section 1.6 lists the papers

[1] The IEEE definition of error, fault and failure is used throughout this thesis.

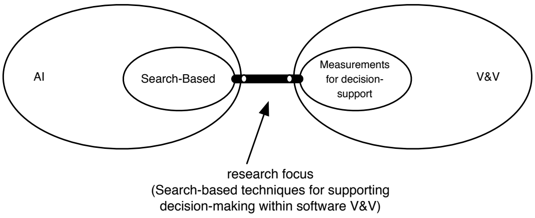

AI
Search-Based
Measurements for decision-support
V&V
research focus
(Search-based techniques for supporting decision-making within software V&V)

Figure 1.1: Cross-connecting concerns addressed in this thesis.

which are part of this dissertation. Section 1.7 provides a short summary of papers that have been published but are not included in this thesis. Finally, Section 1.8 presents a summary of the chapter.

1.2 Concepts and related work

Software engineering data, like any other data, becomes useful only when it is turned into information through analysis. This information can be used to support investigating different themes in this thesis; thus forming a potential decision-support system. Such decisions can ultimately affect scheduling, cost and quality of the end product. However, it is worth keeping in mind that the nature of typical software engineering data[2] is such that different machine learning techniques [20, 57] might be helpful in understanding a rather complex and changing software engineering process. The following subsections describe the concepts and their use in this thesis. We discuss the concepts of software engineering measurement, software quality measurement and search-based software engineering.

1.2.1 Software engineering measurement

The importance of measurement in software engineering is widely acknowledged, especially in helping management in decision-making activities, such as [112]: estimating; planning; scheduling; and tracking.

In formal terms, *measurement is the process by which numbers or symbols are assigned to attributes of entities (e.g., elapsed time in a software testing phase) in the*

[2] Software engineering data is normally characterized by collinearity, noise, large number of inputs and changing input-generating processes.

real world in such a way as to describe them according to clearly defined rules [102, 162]. The numbers or symbols thus assigned are called metrics that signify the degree to which a certain entity possesses a given attribute [302]. The scope of measurement in software engineering can include several activities [102]:

- Cost and effort estimation.
- Productivity measures and models.
- Data collection.
- Quality models and measures.
- Reliability models.
- Performance evaluation and models.
- Structural and complexity metrics.
- Capability-maturity assessment.
- Management by metrics.
- Evaluation of methods and tools.

These activities are supported by a range of software metrics; a common categorization is based on the management function they address, i.e., project, process or product metrics [102, 104]:

- Project metrics — Used on a project level to monitor progress, e.g., number of faults found in integration testing.
- Process metrics — Used to identify the strengths and weaknesses of processes, and to evaluate processes after they have been implemented or changed [141], e.g., system test effort.
- Product metrics — Used to measure and assess the artifacts produced during the software life cycle. Product metrics can further be differentiated into external product metrics and internal product metrics. External product metrics measure what we commonly refer to as quality attributes (behavioral characteristics, e.g., usability, reliability, portability, efficiency). Internal product metrics measure the software attributes itself, e.g., lines of code.

These categories of metrics are related, e.g., a process has an impact on project outcomes. Figure 1.2 depicts that relationship [141].

1.2.2 Software quality measurement

The notion of software quality is not easy to define. There can be a number of desired qualities relevant to a particular perspective of the product, and these can be required

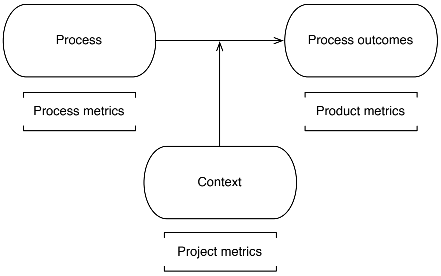


Figure 1.2: The three categories of metrics are related.

to a greater or lesser degree [141].

Software quality metrics focus on measuring the quality of the product, process and project. They can further be divided into end-product quality (e.g., mean time to failure) and in-process (e.g., phase-based fault inflow) quality metrics according to Kan [162]:

> *The essence of software quality engineering is to investigate the relationships among in-process metrics, project characteristics, and end-product quality, and, based on the findings, to engineer improvements in both process and product quality.*

Software quality evaluation models are often applied to aid the interpretation of these relationships. One classification of software quality evaluation models has been presented by Tian [310] and will be discussed briefly in the following subsection.

Tian's classification of quality evaluation models

This section serves as a summary of the classification approach given by Tian [310]. This approach divides the quality evaluation models into two types: generalized models and product-specific models.

Generalized models are not based on project-specific data; rather they take the form of industrial averages. These can further be categorized into three subtypes:

- An overall model. Providing a single estimate of overall product quality, e.g., a single defect density estimate [162].

- A segmented model. Providing quality estimates for different industrial segments, e.g., defect density estimate per market segment.

- A dynamic model. Providing quality estimates over time or development phases, e.g., the Putnam model [274] which generalizes empirical effort and defect profiles over time into a Rayleigh curve[3].

Product-specific models are based on product-specific data. These types of models can also be divided into three subtypes:

- Semi-customized models: Providing quality extrapolations using general characteristics and historical information about the product, process or environment, e.g., a model based on fault-distribution profile over development phases.

- Observation-based models: Providing quality estimates using current project estimations, e.g., various software reliability growth models [223].

- Measurement-driven predictive models: Providing quality estimates using measurements from design and testing processes [325].

Software fault prediction models, a form of quality evaluation models, are of particular relevance for this thesis and are discussed in the next subsection.

Software fault prediction

Errors, faults, failures and defects are inter-related terminologies and often have considerable disagreement in their definitions [100]. However, making a distinction between them is important and therefore for this purpose, we follow the IEEE Standard Glossary of Software Engineering Terminology [301]. According to this, an error is a human mistake, which produces an incorrect result. The manifestation of an error results in a software fault which, in turn, results into a software failure that, translates into an inability of the system or component to perform its required functions within specified requirements. A defect is considered to be the same as a fault [100] although it is a term more common in hardware and systems engineering [301].

In this thesis, the term fault is associated with mistakes at the coding level. These mistakes are found during testing at unit and system levels. Although the anomalies reported during system testing can be termed as failures, we remain persistent with using the term *fault* since it is expected that all the reported anomalies are tracked

[3] Traditionally, a Rayleigh curve indicates the relationship between effort and time-to-market.

down to the coding level. In other words the faults we refer to are pre-release faults, an approach similar to the one taken by Fenton and Ohlsson [101].

Software fault prediction models belong to the family of quality evaluation models. These models are used for objective assessments and problem-area identification [310], thus enabling dual improvements of both product and process. Presence of software faults is usually taken to be an important factor in software quality, a factor that shows generally an absence of quality [134]. A fault prediction model uses previous software quality data in the form of software metrics to predict the number of faults in a component or release of a software system [178]. There are different types of software fault prediction models proposed in software verification and validation literature, all of them with the objective of accurately quantifying software quality. From a holistic point of view, fault-prediction studies can be categorized as making use of traditional (statistical regression) and machine learning (ML) approaches. (The use of machine learning approaches for fault prediction modeling is more recent [354].)

Machine learning is a sub-area within the broader field of artificial intelligence (AI), and is concerned with programming computers to optimize a performance criterion using example data or past experience [20]. Within software engineering predictive modeling, machine learning has been applied for the tasks of classification and regression [354]. The main motivation behind using machine learning techniques is to overcome difficulties in making trustworthy predictions. These difficulties are primarily concerned with certain characteristics that are common in software engineering data. Such characteristics include missing data, large number of variables, heteroskedasticity[4], complex non-linear relationships, outliers and small size of the data sets [113]. Various machine learning algorithms have been applied for software fault prediction; a non-exhaustive summary is provided in Section 4.2 of this thesis. Apart from the classification based on the approaches, Fenton and Neil [100] presents a classification scheme of software fault prediction studies that is based on the different kinds of predictor variables used. The next subsection discusses this classification.

Fenton and Neil's classification of software fault prediction models

Fenton and Neil [100] views the development of software fault prediction models as belonging to four classes:

- Prediction using size and complexity metrics.
- Prediction using testing metrics.
- Prediction using process quality data.

[4] A set of random variables with different variances.

- Multivariate approaches.

Predictions using size and complexity metrics represent the majority of the fault prediction studies. Different size metrics have been used to predict the number of faults, e.g., Akiyama [15] and Lipow [213] used lines of code. There are also studies making use of McCabe's cyclomatic complexity [232], e.g., as in [192]. Then there are studies making use of metrics available earlier in the life cycle, e.g., Ohlsson and Alberg [259] used design metrics to identify fault-prone components.

Prediction using testing metrics involves predicting residual faults by using faults found in earlier inspection and testing phases [54]. Test coverage metrics have also been used to obtain promising results for fault prediction [325].

Prediction using process quality data relates quality to the underlying process used for developing the product, e.g., faults relating to different capability maturity model (CMM) levels [152].

Multivariate approaches to prediction use a small representative set of metrics to form multilinear regression models. Studies report advantages of using such an approach over univariate fault models [175, 244, 245].

1.2.3 Search-based software engineering (SBSE)

Search-based software engineering (SBSE) is a name given to a new field concerned with the application of techniques from metaheuristic search, operations research and evolutionary computation to solve software engineering problems [127, 128, 131]. These computational techniques are mostly concerned with modeling a problem in terms of an evaluation function and then using a search technique to minimize or maximize that function [57]. SBSE treats software engineering problems as a search for solutions that often balances different competing constraints to achieve an optimal or near-optimal result. The basic motivation is to shift software engineering problems from human-based search to machine-based search [127]. Thus the human effort is focussed on guiding the automated search, rather than actually performing the search [127]. Certain problem characteristics warrant the application of search-techniques, which includes a large number of possible solutions (search space) and no known optimal solutions [130]. Other desirable problem characteristics suitable to search-techniques' application include low computational complexity of fitness evaluations of potential solutions and continuity of the fitness function [130].

There are numerous examples of the applications of SBSE spanning over the whole software development life cycle, e.g., requirements engineering [28], project planning [13], software testing [233], software maintenance [38] and quality assessment [39].

1.3 The application of search-based software engineering in this thesis

This thesis has a focus on using search-based techniques to control variation during software V&V activities. Based on measures that support software quality, the thesis investigates multiple related themes (as outlined in Section 1.1). Thus the research questions addressed in this thesis focus on: *i*) a particular problem theme, and *ii*) application of search-based techniques targeting that particular theme. A major portion of this thesis involves the application of software quality evaluation models to help quantify software quality. In relation to Tian's classification of software quality evaluation models [310] and the classification of software fault prediction models by Fenton and Neil [100] (Section 1.2.2), the scope of predictive modeling in this thesis falls in the categories of product-specific quality evaluation models (with respect to Tian's classification) and predictions using testing metrics (with respect to Fenton and Neil's classification). This is shown in Figure 1.3.

As discussed in Section 1.2.2, at a higher level the fault prediction studies can be categorized as making use of statistical regression (traditional) and machine learning (recent) approaches. There are numerous studies making use of machine learning techniques for software fault prediction. Artificial neural networks represents one of the earliest machine learning techniques used for software reliability growth modeling and software fault prediction. Karunanithi et al. published several studies [163, 164, 165, 166, 167] using neural network architectures for software reliability growth modeling. Other examples of studies reporting encouraging results include [3, 19, 86, 117, 118, 135, 169, 177, 182, 184, 293, 311, 312, 313, 314]. Apart from artificial neural networks, some authors have proposed using fuzzy models, as in [60, 61, 297, 323], and support vector machines, as in [316], to characterize software reliability. There are also studies that use a combination of techniques, e.g., [316], where genetic algorithms are used to determine an optimal neural network architecture and [255], where principal component analysis is used to enhance the performance of neural networks. (The use of genetic programming for software fault prediction is further reviewed in Chapter 2 of this thesis.)

In relation to the thesis content, it is useful to discuss some important constituent design elements. This concerns the use of data sets for predictive modeling, statistical hypothesis testing, the use of evaluation measures and the application of systematic literature reviews.

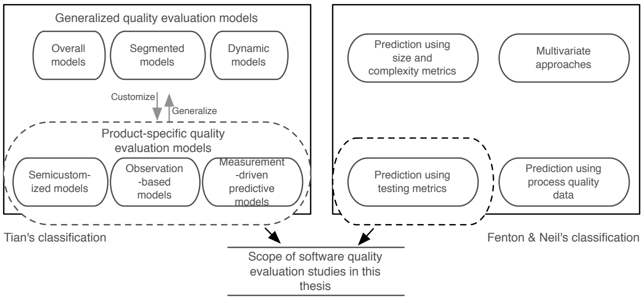

Generalized quality evaluation models
Overall models
Segmented models
Dynamic models
Customize
Generalize
Product-specific quality evaluation models
Semicustomized models
Observation-based models
Measurement-driven predictive models
Tian's classification
Prediction using size and complexity metrics
Multivariate approaches
Prediction using testing metrics
Prediction using process quality data
Fenton & Neil's classification
Scope of software quality evaluation studies in this thesis

Figure 1.3: Relating thesis studies to the two classification approaches.

Software engineering data sets

One of the building blocks of the studies in this thesis is the data sets used for constructing models. A common element in all the data sets used in this thesis is that they come from industry or open source projects. Our industrial partners helped us gather relevant data, while on other occasions, we made use of data from open source software (OSS) projects and repositories such as PROMISE [37]. While the important details of these data sets appear in individual chapters of this thesis, the following points highlight the salient features:

- The data sets are diverse in terms of being both univariate and multivariate. In Chapters 2 and 3, the data sets used resembles a time-series, where occurrences of faults are recorded on weekly/monthly basis. In Chapters 5–8 the data sets are multivariate where multiple metrics related to work progress, test progress and faults found/not found per test phase are used.

- The data sets are diverse in terms of being representative of both closed-source and open-source software projects. Chapter 3 makes use of historical data from three OSS projects while the rest of the chapters make use of industrial, closed-source data sets, made available either by our industrial partners or taken from open-access repositories.

Table 1.1 presents an overview of the data sets used in this thesis. More specific details regarding these data sets are given in relevant chapters.

Table 1.1: Software engineering data sets used in this thesis.

Ch.	Data set names/ description	Source	Context	Characteristics
3	Project 1, Project 2, Project 3	Industrial (through collaboration)	large-scale, telecom	univariate
4	OSStom, OSSbsd, OSSmoz, IND01, IND02, IND03, IND04	Industrial (through collaboration) & open source	large-scale, diverse in domains	univariate & multi-release projects
5	Training project, Testing project	Industrial (through collaboration)	large-scale, telecom	multivariate & on-going projects
6	Training project, Testing project	Industrial (through collaboration)	large-scale, telecom	multivariate & on-going projects
7	AR6, AR1, PC1_req, JM1_req, CM1_req	Industrial (through PROMISE data repository)	large-scale, diverse in domains	multivariate
8	jEdit, AR5, MC1, CM1, KC1_Mod	Industrial (through PROMISE data repository)	large-scale, diverse in domains	multivariate
9	Human resource and bug description data set	Industrial (through collaboration)	large-scale, Enterprise Resource Planning (ERP) software	multivariate

Statistical hypothesis testing and the use of evaluation measures

Statistical hypothesis testing is used to test a formally stated null hypothesis and is a key component in the analysis and interpretation phase of experimentation in software engineering [342]. Earlier studies on predictive accuracy of competing models did not test for statistical significance and, hence, drew conclusions without reporting significance levels. This is, however, not so common anymore as more and more studies report statistical tests of significance[5]. All the chapters in this thesis make use of statistical hypothesis testing to draw conclusions (except for Chapter 2, which is a systematic literature review).

Statistical tests of significance are important since it is not reliable to draw conclusions merely on observed differences in means or medians because the differences could have been caused by chance alone [248]. The use of statistical tests of significance comes with its own share of challenges regarding which tests are suitable for a given problem. A study by Demšar [83] recommends non-parametric (distribution free) tests for statistical comparisons of classifiers; while elsewhere in [48] parametric techniques are seen as robust to limited violations in assumptions and as more powerful (in terms of sensitivity to detect significant outcomes) than non-parametric.

The strategy used in this thesis is to first test the data to see if it fulfills the assumption(s) of a parametric test. If there are no extravagant violations in assumptions,

[5] Simply relying on statistical calculations is not always reliable either, as was clearly demonstrated by Anscombe in [23] where he showed the necessity of actually looking at plotted data.

parametric tests are preferred; otherwise non-parametric tests are used. We are however well aware of the fact that the issue of parametric vs. non-parametric methods is a contentious issue in some research communities. Suffice it to say, if a parametric method has its assumptions fulfilled it will be somewhat more efficient and some non-parametric methods simply cannot be significant on the 5% level if the sample size is too small, e.g., the Wilcoxon signed-rank test [340].

Prior to applying statistical testing, suitable accuracy indicators are required. However, there is no consensus concerning which accuracy indicator is the most suitable for the problem at hand. Commonly used indicators suffer from different limitations [105, 289]. One intuitive way out of this dilemma is to employ more than one accuracy indicator, so as to better reflect on a model's predictive performance in light of different limitations of each accuracy indicator. This way the results can be better assessed with respect to each accuracy indicator and we can better reflect on a particular model's reliability and validity.

However, reporting several measures that are all based on a basic measure, like mean relative error (MRE), would not be useful because all such measures would suffer from common disadvantages of being unstable [105]. For continuos (numeric) prediction, measures for the following characteristics are proposed in [257]: goodness of fit (Kolmogorov-Smirnov test), model bias (U-plot), model bias trend (Y-plot) and short-term predictability (Prequential likelihood). Although providing a thorough evaluation of a model's predictions, this set of measures lacks a suitable one for variable-term predictability. Variable-term predictions are not concerned with one-step-ahead predictions but with predictions in variable time ahead. In [107, 229], average relative error is used as a measure of variable-term predictability.

As an example of applying multiple measures, the study in Chapter 3 uses measures of prequential likelihood, the Braun statistic and adjusted mean square error for evaluating model validity. Additionally we examine the distribution of residuals from each model to measure model bias. Lastly, the Kolmogorov-Smirnov test is applied for evaluating goodness of fit. More recently, analyzing the distribution of residuals is proposed as an alternative measure [193, 289]. It has the convenience of applying significance tests and visualizing differences in absolute residuals of competing models using box plots.

For binary classification studies in this thesis, where the objective is to evaluate the binary classifiers that categorize instances or software components as being either fault-prone (fp) or non fault-prone (nfp), we have used the area under the receiver operating characteristic curve (AUC) [41] as the single scalar means of expected performance. The use of this evaluation measure is further motivated in Chapter 7.

We also see examples of studies in which the authors use a two-prong evaluation strategy for comparing various modeling techniques. They include both quantitative evaluation and subjective qualitative criteria based evaluation because they consider using only quantitative evaluation as an insufficient way to judge a model's output accuracy. Qualitative criterion-based evaluation judges each method based on conceptual requirements [113]. One or more of these requirements might influence model selection. The study in Chapters 3 and 5 presents such qualitative criteria based evaluation, in addition to quantitative evaluation.

Systematic literature review in this thesis

A systematic review evaluates and interprets all available research relevant to a particular research question [188] and, hence, the aim of the systematic review is thus to consolidate all the evidence available in the form of primary studies. Systematic reviews are at the heart of a paradigm called evidence-based software engineering [92, 155, 189], which is concerned with objective evaluation and synthesis of high quality primary studies relevant to a research question. A systematic review differs from a traditional review in the following ways [318]:

- The systematic review methodology is made explicit and open to scrutiny.
- The systematic review seeks to identify *all* the available evidence related to the research question so it represents the totality of evidence.
- The systematic reviews are less prone to selection, publication and other biases.

The guidelines for performing systematic literature reviews in software engineering [188] divides the stages in a systematic review into three phases:

1. Planning the review.
2. Conducting the review.
3. Reporting the review.

The key stages within the three phases are depicted in Figure 1.4 and summarized in the following paragraph:

1. *Identification of the need for a review*—the reasons for conducting the review.
2. *Research questions*—the topic of interest to be investigated e.g., assessing the effect of a software engineering technology.

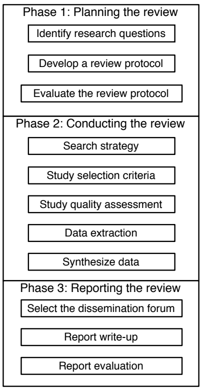

Phase 1: Planning the review
Identify research questions
Develop a review protocol
Evaluate the review protocol
Phase 2: Conducting the review
Search strategy
Study selection criteria
Study quality assessment
Data extraction
Synthesize data
Phase 3: Reporting the review
Select the dissemination forum
Report write-up
Report evaluation

Figure 1.4: The systematic review stages.

3. *Search strategy for primary studies*—the search terms, search query, electronic resources to search, manual search and contacting relevant researchers.

4. *Study selection criteria*—determination of quality of primary studies e.g., to guide the interpretation of findings.

5. *Data extraction strategy*—designing the data extraction form to collect information required for answering the review questions and to address the study quality assessment.

6. *Synthesis of the extracted data*—performing statistical combination of results (meta-analysis) or producing a descriptive review.

Chapter 2 consists of a systematic literature review. This systematic review consolidates the application of symbolic regression using genetic programming (GP) for predictive studies in software engineering.

1.4 Research questions and contribution

The purpose and goals of a research project or study are very often highlighted in the form of specific research questions [77]. These research questions relate to one or more main research question(s) that clarify the central direction behind the entire investigation [77].

The purpose of this thesis, as outlined in Section 1.1, is to evaluate the use of search-based techniques for supporting the decision-making process within software V&V activities, thus impacting the quality of software. The main research question of the thesis is based on this purpose and is formulated as:

> *Main Research Question:* How can search-based techniques be used for improving predictions regarding software quality?

To be able to answer the main research question several other research questions (RQ1–RQ7) need to be answered. In Figure 1.5, the different research questions and how they relate to each other is shown. Figure 1.5 also shows the research process steps relating to RQ1–RQ7 that is further discussed in the upcoming Section 1.5.4.

The first research question that needed an answer, after the main research question was formulated as:

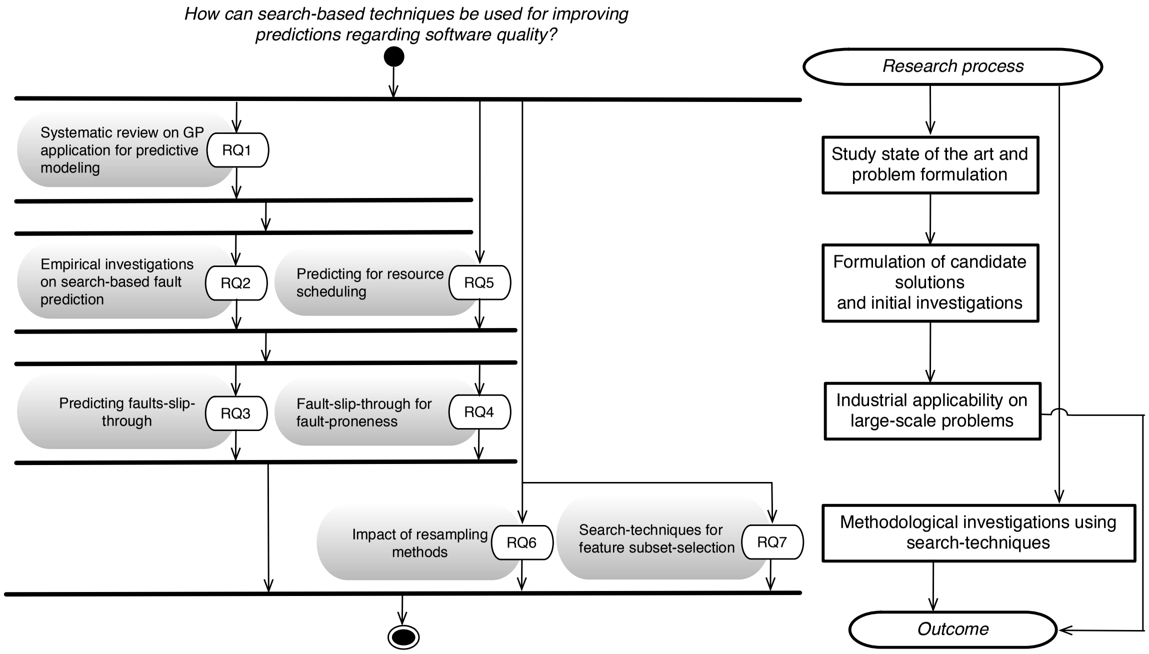
> RQ1: What is the current state of research on using genetic programming (GP) for predictive studies in software engineering?

The answer to RQ1 is to be found in Chapter 2. RQ1 is answered using a systematic literature review investigating the extent of application of symbolic regression in genetic programming within software engineering predictive modeling by:

- Consolidating the available research on the application of GP for predictive modeling studies in software engineering and its performance evaluation by following a systematic process of research identification, study selection, study quality assessment, data extraction and data synthesis.
- Identifying areas of improvement in current studies and highlights further research opportunities.
- Presenting an opportunity to analyze how different improvements/variations to the search mechanism can be transferred from predictive studies in one domain to the other.

Our second research question, RQ2, undertakes initial investigations to apply GP for a particular predictive modeling domain. RQ2 is answered in Chapters 3 and 4 of this thesis.

Figure 1.5: Relationship between different research questions in this thesis.


How can search-based techniques be used for improving predictions regarding software quality?
Systematic review on GP application for predictive modeling
RQ1
Empirical investigations on search-based fault prediction
RQ2
Predicting for resource scheduling
RQ5
Predicting faults-slip-through
RQ3
Fault-slip-through for fault-proneness
RQ4
Impact of resampling methods
RQ6
Search-techniques for feature subset-selection
RQ7
Research process
Study state of the art and problem formulation
Formulation of candidate solutions and initial investigations
Industrial applicability on large-scale problems
Methodological investigations using search-techniques
Outcome

> RQ2: What is the quantitative and qualitative performance of GP in modeling fault count data in comparison with common software reliability growth models, machine learning techniques and statistical regression?

Chapter 3 serves as a stepping-stone for conducting further research in search-based software fault prediction. Chapter 3 helps us answer RQ2 in multiple steps. Specifically, the first step discusses the mechanism enabling GP to progressively search for better solutions and potentially be an effective prediction tool. The second step explores the use of GP for software fault count predictions by evaluating against five different performance measures. This step did not include any comparisons with other models, which were added as a third step in which the predictive capabilities of the GP algorithm were compared against three traditional software reliability growth models.

The early positive results of using GP for fault predictions in Chapter 3 warranted further investigation into this area, resulting in Chapter 4. Chapter 4 investigates *cross-release* prediction of fault data from large and complex industrial and open source software. The complexity of these projects is attributed to being targeting complex functionality, in diverse domains. The comparison groups, in addition to using symbolic regression in GP, include both traditional and machine learning models, while the evaluation is done both quantitatively and qualitatively. Chapters 3 and 4 thus:

- Explore the GP mechanism that might be suitable for modeling.
- Empirically investigate the use of GP as a potential prediction tool in software verification and validation.
- Comparatively evaluate the use of GP with software reliability growth models.
- Evaluate the use of GP for cross-release predictions, for both large-scale industrial and open source software projects.
- Assess GP, both qualitatively and quantitatively, in comparison with software reliability growth models, statistical regression and machine learning techniques for *cross-release* prediction of fault data.

The successful initial investigations in Chapters 3 and 4 encouraged us to make the research results particularly relevant for an industrial setting, where a variety of independent variables play an important role. One of our industrial partners were interested in investigating ways to improve the test phase[6]efficiency. One way to improve test

[6] The test phases are taken in this thesis as synonym to test levels to remain consistent with the company-wide use of terminology. Throughout the thesis, test phases and test levels are used interchangeably.

phase efficiency is to avoid unnecessary rework by finding the majority of faults in the phases where they ought to be found. The faults-slip-through (FST) [79, 80] metric is one way of keeping a check on whether or not a fault slipped through the phase where it should have been found. Our next research question (RQ3) was therefore aimed at predicting this metric for test phase efficiency measurements:

> RQ3: How can we predict FST for each testing phase multiple weeks in advance by making use of data about project progress, testing progress and fault inflow from multiple projects?

The answer to RQ3 is to be found in Chapter 5 of this thesis. The answer to RQ3 also shows a shift from univariate prediction to multivariate prediction, so as to include as much context information as possible in the modeling process by:

- Applying search-based techniques in an industrial context where the amount of rework is being monitored using the FST measure.
- Identifying the test phases with excessive FST inflow; making the basis for following it up with test phase efficiency measurements.

RQ3 was aimed at numeric predictions, therefore this prompted us to investigate the possibility of using the FST metric for *binary* classification, i.e., classifying components as either being fault-prone or non fault-prone. In particular, we investigated the possibility of using the number of faults slipping from unit and function test phases to predict the fault-prone components at the integration and system test phases, which then led us to RQ4.

> RQ4: How can we use FST to predict fault prone software components before integration and system test and what is the resulting prediction performance?

The answer to RQ4 is to be found in Chapter 6, and the chapter also:

- Leverages on collected FST data and project-specific data in the repositories to investigate its use as potential predictors of fault-proneness.
- Provides the basis for early reliability enhancement of fault-prone software components in early test phases for successive releases.

Apart from studying state-of-the-art and doing investigations on large-scale industrial problems, the next two research questions (RQ5 and RQ6) are dedicated to what we call as methodological investigations using search-based techniques that are relevant to the task of software quality measurement. These methodological investigations target the use of resampling methods and feature subset selection methods in software quality measurement, two important design elements in predictive and classification studies in software engineering that lack credible research and recommendations. Thus, RQ5 seeks to investigate the potential impact of resampling methods[7] on software quality classification:

> RQ5: How do different resampling methods compare with respect to predicting fault-prone software components using GP?

The motivation for investigating this question is given in Section 7.1 of Chapter 7 of the thesis and the chapter also:

- Empirically compares five common resampling methods using five publicly available data sets using GP as a software quality classification approach.
- Examines the influence of resampling methods to quantify possible differences.

RQ6 makes up the second research question of our methodological investigations. This time we aim at benchmarking feature subset selection (FSS) methods[8] for software quality classification. For multivariate approaches to predictive studies, much work has concentrated on FSS [100], but very few benchmark studies of FSS methods on data from software projects in industry have been conducted. Also the use of an evolutionary computation method, like GP, has rarely been investigated as a FSS method for software quality classification.

> RQ6: How do different feature subset selection methods compare in predicting fault-prone software components[9]?

The answer to RQ6 is to be found in Chapter 8 and the chapter also:

- Empirically evaluates the use of GP as a feature subset selection method in software quality classification and compares it with competing techniques.

[7] A resampling method is used to draw a large number of samples from the original one and thus to reach an approximation of the underlying theoretical distribution. It is based on repeated sampling within the same data set.

[8] The purpose of FSS is to find a subset of the original features of a data set, such that an induction algorithm that is run on data containing only these features generates a classifier with the highest possible accuracy [195]

[9] The term ‘components’ is taken as a synonym to ‘modules’.

- Demonstrates the relative merits of significant predictor variables.
- Quantifies the use of GP as a potentially valid feature subset selection method.

Up till now, the research questions were centered around predictive and classification studies within software V&V. Our next research question, RQ7, takes a step back from predictive studies and presents another perspective on providing effective decision-support. Using a genetic algorithm (a search-based technique), RQ7 aims to investigate the possibility of effectively scheduling bug[10] fixing tasks to developers and testers, using relevant context information:

> RQ7: How to schedule developers and testers to bug fixing activities taking into account both human properties (skill set, skill level and availability) and bug characteristics (severity and priority) that satisfies different value objectives by using a search-based method such as GA and what is the comparative performance with a baseline method such as a simple hill-climbing?

The answer to RQ7 is to be found in Chapter 9, and the chapter also:

- Takes resource capability and availability into account while triaging and fixing the bugs.
- Uses GA to balance competing constraints of schedule and cost in a quest to reach near optimal resource scheduling.
- Presents an initial bug model and a human resource model to support scheduling.

Table 1.2 lists down the related concept(s) for each chapter along with the research question to be answered.

1.5 Research methodology

Research approaches can usually be classified into quantitative, qualitative and mixed methods [77]. A quantitative approach to research is mainly concerned with investigating cause and effect, quantifying a relationship, comparing two or more groups, use of measurement and observation and hypothesis testing [77]. A qualitative approach to research, on the other hand, is based on theory building relying on human

[10] The term ‘bug’ is taken as a synonym to the IEEE definition of fault. The term ‘bug’ is retained for RQ7 to stay consistent with existing literature on scheduling.

Table 1.2: Related concepts used in thesis chapters for answering each research question.

Research question (RQ)	Related concept(s)	Relevant chapter(s)
RQ1	Software engineering measurement (Section 1.2.1), Software quality measurement (Section 1.2.2), Search-based software engineering (SBSE) (Section 1.2.3)	Chapter 2
RQ2	Software engineering measurement (Section 1.2.1), Software quality measurement (Section 1.2.2), Search-based software engineering (SBSE) (Section 1.2.3)	Chapters 3 & 4
RQ3	Software engineering measurement (Section 1.2.1), Software quality measurement (Section 1.2.2), Search-based software engineering (SBSE) (Section 1.2.3)	Chapter 5
RQ4	Software engineering measurement (Section 1.2.1), Software quality measurement (Section 1.2.2), Search-based software engineering (SBSE) (Section 1.2.3)	Chapter 6
RQ5	Software engineering measurement (Section 1.2.1), Software quality measurement (Section 1.2.2), Search-based software engineering (SBSE) (Section 1.2.3)	Chapter 7
RQ6	Software engineering measurement (Section 1.2.1), Software quality measurement (Section 1.2.2), Search-based software engineering (SBSE) (Section 1.2.3)	Chapter 8
RQ7	Software engineering measurement (Section 1.2.1), Search-based software engineering (SBSE) (Section 1.2.3)	Chapter 9

perspectives. The qualitative approach accepts that there are different ways of interpretation [342]. The mixed methods approach involves using both quantitative and qualitative approaches in a single study.

The below text provides a description of different strategies associated with quantitative, qualitative and mixed method approaches [77]. In the end, the relevant research methods for this thesis are discussed.

1.5.1 Qualitative research strategies

Ethnography, grounded theory, case study, phenomenological research and narrative research are examples of some qualitative research strategies [77].

Ethnography studies people in their contexts and natural settings. The researcher usually spends longer periods of time in the research setting by collecting observational data [77]. *Grounded theory* evolved as an abstract theory of the phenomenon under interest based on the views of the study participants. The data collection is continuous and information is refined as progress is made [77]. A *case study* involves in-depth investigation of a single case, e.g., an event or a process. The case study has time and scope delimitations within which different data collection procedures are applied [77]. *Phenomenological research* is grounded in understanding the human experiences concerning a phenomenon [77]. Like in ethnography, phenomenological research involves prolonged engagement with the subjects. *Narrative research* is akin to retelling stories about other individuals' lives while relating to the researcher's life in some manner [77].

1.5.2 Quantitative research strategies

Quantitative research strategies can be divided into two quantitative strategies of inquiry [77]: Experiments and surveys.

An experiment, or "[...] a formal, rigorous and controlled investigation" [342], has as a main idea to distinguish between a control situation and the situation under investigation. Experiments can be true experiments and quasi-experiments. Within quasi-experiment, there can also be a single-subject design.

In a *true experiment*, the subjects are randomly assigned to different treatment conditions. This ensures that each subject has an equal opportunity of being selected from the population; thus the sample is representative of the population [77]. *Quasi-experiments* involve designating subjects based on some non-random criteria. This sample is a convenience sample, e.g., because the investigator must use naturally formed groups. *The single-subject designs* are repeated or continuos studies of a single process or individual. *Surveys* are conducted to generalize from a sample to a population by conducting cross-sectional and longitudinal studies using questionnaires or structured interviews for data collection [77].

Robson, in his book *Real World Research* [281], identifies another quantitative research strategy named *non-experimental fixed designs*. These designs follow the same general approach as used in experimental designs but without active manipulation of the variables. According to Robson, there are three major types of non-experimental fixed designs: relational (correlational) designs, comparative designs and longitudinal designs. First, relational (correlational) designs analyze the relationships between two or more variables and can further be divided into cross-sectional designs and prediction studies. Cross-sectional designs are normally used in surveys and involves in taking measures over a short period of time, while prediction studies are used to investigate if one or more predictor variables can be used to predict one or more criterion variables. Since prediction studies collect data at different points in time, the study extends over time to test these predictions. Second, comparative designs involve analyzing the differences between the groups; while, finally, longitudinal designs analyze trends over an extended period of time by using repeated measures on one or more variables.

1.5.3 Mixed method research strategies

The mixed method research strategies can use sequential, concurrent or transformative procedures [77]. The *sequential* procedure begins with a qualitative method and follows it up with quantitative strategies. This can conversely start with a quantitative method and later on complemented with qualitative exploration [77]. *Concurrent* procedures involve integrating both quantitative and qualitative data at the same time; while *transformative* procedures include either a sequential or a concurrent approach containing both quantitative and qualitative data, providing a framework for topics of interest [77].

With respect to specific research strategies, surveys and case studies can be both quantitative and qualitative [342]. The difference is dependent on the data collection mechanisms and how the data analysis is done. If data is collected in such a manner that statistical methods are applicable, then a case study or a survey can be quantitative.

We consider systematic literature reviews (Section 1.3) as a form of survey. A systematic literature review can also be quantitative or qualitative depending on the data synthesis [188]. Using statistical techniques for quantitative synthesis in a systematic review is called meta-analysis [188]. However, software engineering systematic literature reviews tend to be qualitative (i.e., descriptive) in nature [44]. One of the reason for this is that the experimental procedures used by the primary studies in a systematic literature review differs, making it virtually impossible to undertake a formal meta-analysis of the results [191].

1.5.4 Research methodology in this thesis

The research process used in this thesis is shown in Figure 1.5. We strived for contributions on two fronts: *i*) industrial relevance of the obtained research results, and *ii*) methodological investigations aimed at improving the design of predictive modeling studies.

The research path taken towards the strive for industrial relevance was composed of a multi-step process where answers to different research questions were pursued:

- Studying state-of-the-art and problem formulation (RQ1).
- Formulating candidate solutions and carrying out initial investigations (RQ2 and RQ5).
- Assessing industrial applicability on large-scale problems (RQ3 and RQ4).

The second front of our research — the methodological investigations — included RQ6 and RQ7. These investigations were targeted at two important design elements of predictive modeling studies in software engineering (resampling and feature subset selection).

The research methodology used in this thesis is both quantitative and qualitative. Chapters 3 to 7 of this thesis fall within the category of prediction studies (Section 1.5.2) and thus belonging to the high-level category of non-experimental fixed designs. Specifically, these chapters make use of one or more predictor (independent) variables to predict the criterion (dependent) variable. Also these studies use quantitative data collected over time, which is used for training and testing the models. Chapters 4 and 5 are additionally complemented with a qualitative assessment of models so it is justifiable to place it under a mixed methods approach using sequential procedure. Chapter 2 is a systematic review and since it consists of descriptive data synthesis, it is considered to be a candidate for qualitative studies. Chapters 8 and 9 are comparative design studies analyzing the differences between the groups. Table 1.3 presents the research methodologies used in this thesis in tabular form.

Table 1.3: Research methodologies used in this thesis.

Chapter	Utilized research methodology
2	Qualitative → Survey → Systematic review
3	Quantitative → Non-experimental fixed designs → Relational design → Predictive studies
4	Mixed method → Sequential procedure
5	Mixed method → Sequential procedure
6	Quantitative → Non-experimental fixed designs → Relational design → Predictive studies
7	Quantitative → Non-experimental fixed designs → Relational design → Predictive studies
8	Quantitative → Non-experimental fixed designs → Comparative designs
9	Quantitative → Non-experimental fixed designs → Comparative designs

1.6 Papers included in this thesis

The introductory Chapter 1 is an extended version of a summary paper — *Search-based prediction of fault count data* — published in the proceedings of the *1st International Symposium on Search Based Software Engineering (SSBSE'09)*.

Chapter 2 is based on a systematic literature review — *On the application of genetic programming for software engineering predictive modeling: A systematic review* — accepted at the Journal of Expert Systems with Applications.

Chapter 3 is based on three papers — *Suitability of genetic programming for software reliability growth modeling* — published in the proceedings of the *2008 IEEE International Symposium on Computer Science and its Applications (CSA'08)*, *Prediction of fault count data using genetic programming* — published in the proceedings of the *12th IEEE International Multitopic Conference (INMIC'08)* and *A comparative evaluation of using genetic programming for predicting fault count data* — published in the proceedings of the *3rd International Conference on Software Engineering Advances (ICSEA'08).*

Chapter 4 has been published as a book chapter with the title — *Genetic programming for cross-release fault count predictions in large and complex software projects*, in the book — *Evolutionary computation and optimization algorithms in software engineering* — published by IGI Global.

Chapter 5 is based on an extended version of the research paper — *Search-based prediction of faults-slip-through in large software projects* — published in the proceedings of the *2nd International Symposium on Search Based Software Engineering (SSBSE'10).* The extended version is under submission at *IEEE Transactions on Software Engineering*.

Chapter 6 is based on the research paper — *Using faults-slip-through metric as a predictor of fault-proneness* — published in the proceedings of the — *17th Asia Pacific Software Engineering Conference (APSEC'10).*

Chapter 7 — *Resampling methods in software quality classification – A comparison using genetic programming* — has been submitted to the *International Journal of Software Engineering and Knowledge Engineering*, special issue on: *Emerging synergies of artificial intelligence and software engineering*.

Chapter 8 — *Genetic programming for feature subset selection – A comparative evaluation* — is based on the manuscript — *Benchmarking feature subset selection methods for software fault prediction*, that has been submitted to the *Journal of Systems and Software*.

Chapter 9 is based on the research paper — *Search-based resource scheduling for bug-fixing tasks* — published in the proceedings of the *2nd International Symposium on Search Based Software Engineering (SSBSE'10).*

Wasif Afzal is first author of all papers except the paper that Chapter 9 is based on (a first author has the main responsibility for the idea, implementation, conclusion and composition of the results). Chapter 9 was written together with Dr. Junchao Xiao. Wasif Afzal was involved in the setup of the experiment as well as investigating and drawing conclusions on the empirical part. He also took part in the theoretical discussions and in writing the paper.

Dr. Richard Torkar is a co-author on all the papers included in this thesis except the papers that Chapters 6 and 9 are based on. Dr. Robert Feldt is a co-author on the papers that Chapters 3–5 and 7 are based on. Dr. Tony Gorschek is a co-author on the papers that Chapters 4 and 5 are based on. Finally, Dr. Greger Wikstrand is a co-author on the paper that Chapter 5 is based on.

1.7 Papers also published but not included

Except for the papers included in this thesis, a number of related additional papers have also been published. The following papers are, however, not included in this thesis:

Paper I W. Afzal, R. Torkar and R. Feldt. A systematic review of search-based testing for non-functional system properties. Information and Software Technology, Volume 51, Issue 6, 2009.

Paper II R. Torkar, N. M. Awan, A. K. Alvi and W. Afzal. Predicting software test effort in iterative development using a dynamic bayesian network. Proceedings of the 21st International Symposium on Software Reliability Engineering (ISSRE'10) – Industry track.

Paper III W. Afzal and R. Torkar. Lessons from applying experimentation in software engineering prediction systems. Proceedings of The 2nd International Workshop on Software Productivity Analysis and Cost Estimation (SPACE'08), Collocated with 15th Asia-Pacific Software Engineering Conference (APSEC'08).

Paper IV W. Afzal and R. Torkar. Incorporating metrics in an organizational test strategy. Proceedings of the International Software Testing Standard Workshop, Collocated with 1st International Conference on Software Testing, Verification and Validation (ICST'08).

Paper V R. Feldt, R. Torkar, T. Gorschek and W. Afzal. Searching for cognitively diverse tests: Towards universal test diversity metrics. Proceedings of the 1st International Workshop on Search-based Software Testing (SBST'08), Collocated with 1st International Conference on Software Testing, Verification and Validation (ICST'08).

1.8 Summary

In this chapter we presented a synopsis of the research area and what we believe to be the main contributions of the research presented in this thesis. We additionally presented the concepts that will be used in later chapters and outlined the research methodology used. The next chapter, Chapter 2, presents a systematic literature review on the applicability of genetic programming as a predictive modeling tool in software engineering.

Chapter 2

On the application of genetic programming for software engineering predictive modeling: A systematic review

Accepted for publication in the
Journal of Expert Systems with
Applications

W. Afzal & R. Torkar

2.1 Introduction

Genetic programming (GP) [200] is an evolutionary computation technique. It is a *systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done* [272]. Symbolic regression is one of the many application areas of GP, which finds a function with the outputs having desired outcomes. It has the advantage of being independent of making any assumptions about the function structure. Another potential advantage is that models built using symbolic regression application of GP can also help in identifying the significant variables which might be used in subsequent modeling attempts [198]. This chapter reviews the available literature on the application of symbolic regression using GP for predictions and estimations within software engineering. The performance of symbolic regression using GP is assessed in terms of its comparison with competing models, which might include common machine learning models, statistical models and models based on expert opinion. There are two reasons for carrying out this study:

1. To be able to draw (if possible) general conclusions about the extent of application of symbolic regression using GP for predictions and estimations in software engineering.

2. To summarize the benefits and limitations of using symbolic regression as a prediction and estimation tool.

The authors are not aware of any study having goals similar to ours. Prediction and estimation in software engineering has been applied to measure different attributes. A non-exhaustive list includes prediction and estimation of software quality, e.g., [204], software size, e.g., [220], software development cost/effort, e.g., [157], maintenance task effort, e.g., [153], correction cost, e.g., [82], software fault, e.g., [307], and software release timing, e.g., [86]. A bulk of the literature contributes to software cost/ effort and software fault prediction. A systematic review of software fault prediction studies is given by Catal and Diri [66], while a systematic review of software development cost estimation studies is provided by [157]. This chapter differs from these systematic reviews in several ways. Firstly, the studies of [66] and [157] are more concerned with classification of primary studies and capturing different trends. This is not the primary purpose of this study, which is more concerned with investigating the comparative efficacy of using symbolic regression across software engineering predictive studies. Secondly, [66] and [157] review the subject area irrespective of the applied method, resulting in being more broad in their coverage of the specific area. This is not the case with this study as it is narrowly focused in terms of the applied technique and open in terms of capturing prediction and estimation of different attributes (as will be evident from the addressed research question in Section 2.2.1). Thirdly, one additional concern, which makes this study different from [66] and [157], is that it also assesses the evidence of comparative analysis of symbolic regression with other competing models.

A paper by Crespo et al. [76] presents a classification of software development effort estimation into artificial intelligence (AI) methods of neural networks, case-based reasoning, regression trees, fuzzy logic, dynamical agents and genetic programming. While the authors were able to present a classification scheme, it is not complete in terms of its coverage of studies within each AI method.

One other motivation for carrying out this systematic review is the general growing interest in search-based approaches to solve software engineering problems [131]. In this regard, it is interesting to investigate the extent of application of genetic programming (a search-technique) within software engineering predictive modeling. This presents an opportunity to assess different attributes, which can be measured using GP. It also allows us to gain an understanding of different GP variations used by these studies to predict and estimate in a better way.

In the rest of this chapter, wherever we refer to GP, we mean the symbolic regression application of GP.

This chapter is organized as follows: Section 2.2 describes the research method including the research question, the search strategy, the study selection procedure, the study quality assessment and the data extraction. Results are presented in Section 2.3, while Section 2.4 discusses the results and future work. Validity threats and conclusions appear in Section 2.5 and Section 2.6, respectively.

2.2 Method

This section describes our review protocol, which is a multi-step process following the guidelines outlined in [188].

2.2.1 Research question

We formulated the following research question for this study:

RQ Is there evidence that symbolic regression using genetic programming is an effective method for prediction and estimation, in comparison with regression, machine learning and other models?

The research questions can conveniently be structured in the form of PICOC (Population, Intervention, Comparison, Outcome, Context) criteria [268]. The *population* in this study is the domain of software projects. *Intervention* includes models evolved using symbolic regression application of GP. The *comparison* intervention includes the models built using regression, machine learning and other methods. The *outcome* of our interest represents the comparative effectiveness of prediction/estimation using symbolic regression and machine learning/regression/other models. We do not pose any restrictions in terms of context and experimental design.

2.2.2 The search strategy

Balancing comprehensiveness and precision in the search strategy is both an important and difficult task. We used the following approach for minimizing the threat of missing relevant studies:

1. *Breaking down the research question into PICOC criteria.* This is done to manage the complexity of a search string that can get rather sophisticated in pursuit of comprehensiveness.
2. *Identification of alternate words and synonyms for each of PICOC criterion.* First, since it is common that terminologies differ in referring to the same concept, derivation of alternate words and synonyms helps ensuring completeness of search. The genetic programming bibliography maintained by Langdon et al. [202] and Alander's bibliography of genetic programming [16] turned out to be valuable sources for deriving the alternate words and synonyms. Secondly our experience of conducting studies in a similar domain was also helpful [10].
3. *Use of Boolean `OR` to join alternate words and synonyms.*
4. *Use of Boolean `AND` to join major terms.*

We came up with the following search terms (divided according to the PICOC criteria given in Section 2.2.1):

- **Population.** software, application, product, web, internet, world wide web, project, development.
- **Intervention.** symbolic regression, genetic programming.
- **Comparison intervention.** regression, machine learning, machine-learning, model, modeling, modelling, system identification, time series, time-series.
- **Outcomes.** prediction, assessment, estimation, forecasting.

Hence, leading to the following search string: (software OR application OR product OR Web OR Internet OR "World-Wide Web" OR project OR development) AND ("symbolic regression" OR "genetic programming") AND (regression OR "machine learning" OR machine-learning OR model OR modeling OR modelling OR "system identification" OR "time series" OR time-series) AND (prediction OR assessment OR estimation or forecasting).

The search string was applied to the following digital libraries, while searching within all the available fields (i.e., abstract, titles, key words, etc.):

- INSPEC
- EI Compendex
- ScienceDirect
- IEEEXplore
- ISI Web of Science (WoS)
- ACM Digital Library

In order to ensure the completeness of the search strategy, we compared the results with a small core set of primary studies we found relevant, i.e., [56, 73, 88]. All of the known papers were found using multiple digital libraries.

We additionally scanned the online GP bibliography maintained by Langdon et al. [202] by using the search-term *symbolic regression*. We also searched an online data base of software cost and effort estimation called BESTweb [154], using the search-term *genetic programming*.

The initial automatic search of publication sources was complemented with manual search of selected journals (J) and conference proceedings (C). These journals and conference proceedings were selected due to their relevance within the subject area and included: Genetic Programming and Evolvable Machines (J), European Conference on Genetic Programming (C), Genetic and Evolutionary Computation Conference (C), Empirical Software Engineering (J), Information and Software Technology (J), Journal of Systems and Software (J), IEEE Transactions on Software Engineering (J) and IEEE Transactions on Evolutionary Computation (J). We then also scanned the reference lists of all the studies gathered as a result of the above search strategy to further ensure a more complete set of primary studies.

The time span of the search had a range of 1995–2008. The selection of 1995 as the starting year was motivated by the fact that we did not find any relevant study prior to 1995 from our search of relevant GP bibliographies [16, 202]. In addition, we also did not find any relevant study published before 1995 as a result of scanning of the reference lists of studies found by searching the electronic databases.

2.2.3 The study selection procedure

The purpose of the study selection procedure is to identify primary studies that are directly related to answering the research question [188]. We excluded studies that:

1. Do not relate to software engineering or software development, e.g., [18].

2. Do not relate to prediction/estimation of software cost/effort/size, faults, quality, maintenance, correction cost and release timing, e.g., [2].

3. Report performance of a particular technique/algorithmic improvement without being applied to software engineering, e.g., [22].

4. Do not relate to symbolic regression (or any of its variants) using genetic programming, e.g., [291].

5. Do not include a comparison group, e.g., [172].

6. Use genetic programming only for feature selection prior to using some other technique, e.g., [277].

7. Represent similar studies, i.e., when a conference paper precedes a journal paper. As an example, we include the journal article by Costa et al. [73] but exclude two of theirs conference papers [75, 262].

Table 2.1 presents the count of papers and the distribution before and after duplicate removal as a result of the automatic search in the digital libraries.

Table 2.1: Count of papers before and after duplicate removal for the digital search in different publication sources. The numbers within parenthesis indicates the counts after duplicate removal.

Source	Count
EI Compendex & Inspec	578 (390)
ScienceDirect	496 (494)
IEEE Xplore	55 (55)
ISI Web of Science	176 (176)
ACM Digital Library	1081 (1081)
Langdon et al. GP bibliography [202]	342 (342)
BESTweb [154]	4 (4)
Total	2732 (2542)

The exclusion was done using a multi-step approach. First, references were excluded based on title and abstract which were clearly not relevant to our research question. The remaining references were subject to a detailed exclusion criteria (see above) and, finally, consensus was reached among the authors in including 24 references as primary studies for this review.

2.2.4 Study quality assessment and data extraction

The study quality assessment can be used to devise a detailed inclusion/exclusion criteria and/or to assist data analysis and synthesis [188]. We did not rank the studies according to an overall quality score but used a simple 'yes' or 'no' scale [91]. Table 1, Appendix A (page 281), shows the application of the study quality assessment criteria where a ($\surd$) indicates 'yes' and ($\times$) indicates 'no'. Further a (~$\surd$) shows that we were not sure as not enough information was provided but our inclination is towards 'yes' based on reading full text. A (~$\times$) shows that we were not sure as not enough information was provided but our inclination is towards 'no' based on reading full text. We developed the following study quality assessment criteria, taking guidelines from [188, 191]:

- Are the aims of the research/research questions clearly stated?
- Do the study measures allow the research questions to be answered?
- Is the sample representative of the population to which the results will generalize?
- Is there a comparison group?
- Is there an adequate description of the data collection methods?
- Is there a description of the method used to analyze data?
- Was statistical hypothesis undertaken?
- Are all study questions answered?
- Are the findings clearly stated and relate to the aims of research?
- Are the parameter settings for the algorithms given?
- Is there a description of the training and testing sets used for the model construction methods?

The data extraction was done using a data extraction form for answering the research question and for data synthesis. One part of the data extraction form included the standard information of title, author(s), journal and publication detail. The second part of the form recorded the following information from each primary study: stated hypotheses, number of data sets used, nature of data sets (public or private), comparison group(s), the measured attribute (dependent variable), evaluation measures used, independent variables, training and testing sets, major results and future research directions.

2.3 Results

The 24 identified primary studies were related to the prediction and estimation of the following attributes:

1. Software fault proneness (software quality classification).
2. Software cost/effort/size (CES) estimation.
3. Software fault prediction and software reliability growth modeling.

Table 2.2 describes the relevant information regarding the included primary studies. The 24 primary studies were related to the application of GP for software quality classification (9 primary studies), software CES estimation (7 primary studies) and software fault prediction and software reliability growth modeling (8 primary studies).

Figure 2.1 shows the year-wise distribution of primary studies within each category as well as the frequency of application of the different comparison groups. The bubble at the intersection of axes contains the number of primary studies. It is evident from the left division in this figure that the application of GP to prediction problems in software engineering has been scarce. This finding is perhaps a little surprising; considering that the proponents of symbolic regression application of GP have highlighted several advantages of using GP for solving prediction problems [203].

In the right division of Figure 2.1, it is also clear that statistical regression techniques (linear, logistic, logarithmic, cubic, etc.) and artificial neural networks have been used as a comparison group for most of the studies.

Next we present the description of the primary studies in relation to the research question.

2.3.1 Software quality classification

Our literature search found 10 studies on the application of symbolic regression using GP for software quality classification. Seven out of these ten studies were co-authored by similar authors to a large extent, where one author was found to be part of each of these seven studies. The data sets also overlapped between studies, which gives an indication that the conclusion of these studies were tied to the nature of the data sets used. However, these seven studies were marked with different variations of the GP fitness function and also used different comparison groups. This in our opinion indicates a distinct contribution and thus worthy of inclusion as primary studies for this

Table 2.2: Distribution of primary studies per predicted/estimated attribute.

Author(s)	Year	Ref.	Domain
Robinson et al.	1995	[280]	SW quality classification (37.50%)
Evett et al.	1998	[96]	
Khoshgoftaar et al.	2003	[180]	
Liu et al.	2001	[215]	
Khoshgoftaar et al.	2004	[173]	
Khoshgoftaar et al.	2004	[174]	
Liu et al.	2004	[216]	
Reformat et al.	2003	[278]	
Liu et al.	2006	[218]	
Dolado et al.	1998	[89]	SW CES estimation (29.17%)
Dolado	2000	[87]	
Regolin et al.	2003	[279]	
Dolado	2001	[88]	
Burgess et al.	2001	[56]	
Shan et al.	2002	[288]	
Lefley et al.	2003	[208]	
Kaminsky et al.	2004	[160]	SW fault prediction and reliability growth (33.33%)
Kaminsky et al.	2004	[161]	
Tsakonas et al.	2008	[322]	
Zhang et al.	2006	[355]	
Zhang et al.	2008	[356]	
Afzal et al.	2008	[6]	
Costa et al.	2007	[73]	
Costa et al.	2006	[74]	

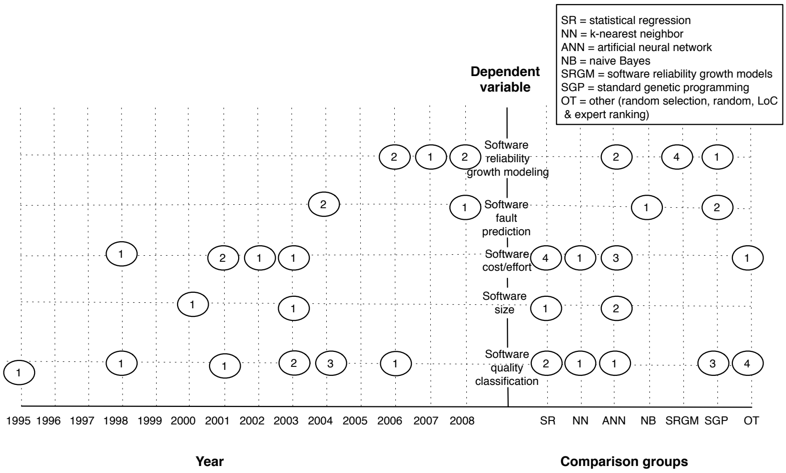


Figure 2.1: Distribution of primary studies over range of applied comparison groups and time period.

review. The evaluation measures also varied but were mostly based on the Type-I and Type-II misclassification rates.

A software quality classification model predicts the fault-proneness of a software component as being either fault-prone (*fp*) or not fault-prone (*nfp*). A fault-prone component is one in which the number of faults are higher than a selected threshold. The use of these models leads to knowledge about problematic areas of a software system, that in turn can trigger focused testing of fault-prone components. With limited quality assurance resources, such knowledge can potentially yield cost-effective verification and validation activities with high return on investment.

The general concept of a software quality classification model is that it is built based on the historical information of software metrics for program components with known classification as fault-prone or not fault-prone. The generated model is then tested to predict the risk-based class membership of components with known software metrics in the testing set.

Studies making use of GP for software quality classification argue that GP carries certain advantages for quality classification in comparison with traditional techniques because of its white-box and comprehensible classification model [180]. This means that GP models can potentially show the significant software metrics affecting the quality of components. Additionally, by following a natural evolution process, GP can automatically extract the underlying relationships between the software metrics and the software quality, without relying on the assumption of the form and structure of the model.

In [280], the authors use GP to identify fault-prone software components. A software component is taken to comprise of a single source code file. Different software metrics were used as independent variables, with predictions assessed using five and nine independent variables. GP was compared with neural networks, *k*-nearest neighbor and linear regression. The methods were compared using two evaluation measures, accuracy and coverage. Accuracy was defined as the proportion of 'files predicted to be faulty' which were faulty, while coverage was defined as the proportion of 'files which were faulty' which were accurately predicted to be faulty. Using a measurement data corresponding to 163 software files, it was observed that in comparison with other techniques, GP results were reasonably accurate but lacked coverage.

In [96] the authors describe a GP-based system for targeting software components for reliability enhancement. This study not only predicted the number of faults but also ranked-order the software components. The motivation was to assist the project managers in deciding which software components were more fault-prone. The authors claimed the study to be the first one that applied GP on software quality predictions. However, we found Robinson and McIlroy's study [280] to be the earliest using GP for software quality classification. Using the actual data from two industrial data sets (a data communication system and a legacy telecommunication system), Evett et al. showed that for rank order of components from least to the most fault-prone, the GP models were able to reveal faults closer to the actual number in comparison with random selection of components for reliability enhancement. With cut-off percentile values of 75%, 80%, 85% and 90% for component ordering, GP model performance was consistently superior to random ordering of components based on the number of faults. The problem was solved as a multi-objective optimization problem with minimization of absolute errors in prediction of faults as well as maximization of the best percentage of the actual faults averaged over the percentile level of interest.

A similar approach was used by Khoshgoftaar et al. [173], in which a different multi-objective fitness value: (*i*) Maximized the best percentage of the actual faults averaged over the percentile level of interest (95%, 90%, 80%, 70%). (*ii*) Restricted the size of the GP tree. The data set used in the study came from an embedded software system and five software metrics were used for quality prediction. The data set was divided into three random splits of the training and the testing data sets to avoid biased results. Based on the comparison of models ranked according to lines of code (LoC), the GP-models ranked the components closer to the actual ranking on two of the three data splits. The results were not much different in an extension of this study [174], where in an additional case study of a legacy telecommunication system with 28 independent variables, GP outperformed the component ranking based on LoC.

Another study by Khoshgoftaar et al. [180] used a different multi-objective fitness function for generating the software quality model. First the average weighted cost of misclassification was minimized and subsequently the trees were simplified by controlling their size. The average weighted cost of misclassification was formulated to penalize Type-II errors (a *fp* component misclassified as *nfp*) more than Type-I errors (a *nfp* component misclassified as *fp*). This was done by normalizing the cost of Type-II error with respect to the cost of Type-I errors. Data was collected from two embedded systems applications, which consisted of five different metrics for different components. In comparison with standard GP, the performance of multi-objective GP was found to be better with multi-objective GP finding lower Type-I and Type-II error rates with smaller tree sizes. A similar study was carried out by [215] in which a single objective fitness function was used that took into account the average weighted cost of misclassification. Random subset selection was chosen which evaluated GP individuals in each generation on a randomly selected subset of the fit data set. Random subset selection helped to reduce the problem of over-fitting in GP solutions. Comparisons with logistic regression showed that Type-I and Type-II error rates for GP model were found to be better than for logistic regression. The same authors extended the study by adding a case study with data from a legacy telecommunication system in [216]. This time the fitness function was multi-objective with minimization of expected cost of misclassification and also control of the tree size of GP solutions. The results of applying the random subset selection showed that over-fitting was reduced in comparison with when there was no random subset selection, hence, yielding solutions with better generalizability in the testing part of the data set.

In [278], evolutionary decision-trees were proposed for classifying software objects. The comparison group in this case was the classification done by two architects working on the project under study. The data set consisted of 312 objects whose quality was ranked by two architects as high, medium and low. The independent variables included 19 different software metrics for each object. Both genetic algorithms and GP were used to get the optimal splitting of attribute domains for the decision-tree and to get the best decision-tree. The GA chromosome was represented by a possible splitting for all attributes. The fitness of the chromosome was evaluated using GP with two possibilities of the fitness function: (*i*) When the number of data samples in each class was comparable, $\frac{K}{N}$, where $K =$ number of correctly specified data and $N =$ number of data samples in a training set. (*ii*) When the number of data samples in each class were not comparable, $\prod_{i=1}^{c} \frac{k_i+1}{n_i}$, where $c =$ number of different classes, $n_i =$ number of data samples belonging to a class i and, finally, $k_i =$ number of correctly classified data of a class i. The results showed that in comparison with architects' classification of objects' quality, the rate of successful classification for training data was around 66–72% for the first and the second architect respectively.

In [218], the performance of GP based software quality classification is improved by using a multi data set validation process. In this process, the hundred best models were selected after training on a single data set. These models were then validated on five *validation* data sets. The models that performed the best on these validation data sets were applied to the testing data set. The application of this technique to seven different NASA software projects showed that the misclassification costs were reduced in comparison with standard genetic programming solution.

Tables 2.3 and 2.4[1] shows the relevant summary data extracted to answer the research question from each of the primary studies within the area of software quality classification.

2.3.2 Software cost/effort/size (CES) estimation

In line with what Jørgensen and Shepperd suggest in [157], we will use the term 'cost' and 'effort' interchangeably since effort is a typical cost driver in software develop-

[1] The data sets in Table 2.4 are taken at a coarser level, e.g., ISBSG data ([143]) of multiple projects is one data set.

Table 2.3: Summary data for primary studies on GP application for software quality classification. (?) indicates absence of information and (~) indicates indifferent results.

Fitness function	Comparison group	Evaluation measures	GP better?	Dependent variable	Article
Minimize root mean square	Neural networks, k-nearest neighbor, linear regression	Accuracy & coverage	~	Fault proneness based on number of faults	[280]
Minimize absolute errors, maximize best %age of faults averaged over the percentile level	Random rank ordering	Ranking on the basis of faults in different percentile ranges	√	Fault proneness based on number of faults	[96]
Minimize avg. cost of misclassification and minimize tree size	Standard GP	Type I, Type II and overall error rates	√	Fault proneness based on number of faults	[180]
Minimize avg cost of misclassification	Logistic regression	Type I, Type II and overall error rates	√	Fault proneness based on number of faults	[215]
Maximize best %age of actual faults averaged over the percentiles level of interest, control tree size	Ranking based on lines of code	Number of faults accounted by different cut-off percentiles	√	Number of faults for each software component	[173]
Maximize best %age of actual faults averaged over the percentiles level of interest, control tree size	Ranking based on lines of code	Number of faults accounted by different cut-off percentiles	√	Number of faults for each software component	[174]
Minimize expected cost of misclassification, control tree size	Standard GP	Number of over-fitting models and Type I, Type II error rates	√	Fault proneness based on number of faults	[216]
(a) $\frac{K}{N}$ (b) $\prod_{i=1}^{c} \frac{k_i+1}{n_i}$	Quality ranking of an object assessed by the architects	Rate of successful classification for training and testing set	~	Ranking of object's quality	[278]
Minimize expected cost of misclassification, control tree size	Standard GP	Type I and Type II error rates	√	Fault proneness based on number of faults	[218]

Table 2.4: Data set characteristics for primary studies on GP application for software quality classification. (?) indicates absence of information.

Article	Data sets no.	Sampling of training and testing sets	Industrial (I) or academic (A)	Data sets public or private
[280]	1	103 records for training and 60 records for testing	?	Private
[96]	2	$\frac{2}{3}$ components for training and the rest for testing	I	Private
[180]	1	Approximately $\frac{2}{3}$ for training and the rest for testing	I	Private
[215]	1	Approximately $\frac{2}{3}$ for training and the rest for testing and random subset selection	I	Private
[173]	1	$\frac{2}{3}$ for training and the rest for testing, three splits	I	Private
[174]	2	$\frac{2}{3}$ for training and the rest for testing, three splits	I	Private
[216]	1	Training on release 1 data set, testing on release 2,3,4 data sets	I	Private
[278]	1	10 fold cross-validation	I	Private
[218]	7	1 training data set, 5 validation data sets and 1 testing data set	I	Private

ment. We additionally take software size estimation to be related to either effort or cost and discuss these studies in this same section. According to Crespo et al. [76], six different artificial intelligence methods are common in software development effort estimation. These are neural networks, case-based, regression trees, fuzzy logic, dynamical agents and genetic programming. We are here concerned with the application of symbolic regression using genetic programming as the base technique.

In [89], five different data sets were used to estimate software effort with line of code (LoC) and function points as the independent variables. Using the evaluation measures of $\text{pred}(0.25)$ and MMRE (mean magnitude of relative error), it was observed that with respect to predictive accuracy, no technique was clearly superior. However, neural networks and GP were found to be flexible approaches as compared with classical statistics.

In [87], different hypotheses were tested for estimating the size of the software in terms of LoC. Specifically, the component-based method was validated using three different techniques of multiple linear regression, neural networks and GP. Three different components were identified which included menus, input and reports. The independent variables were taken to be the number of choices within the menus and the number of data elements and relations for inputs and reports. For evaluating the component-based methodology in each project, six projects were selected having largest independent variables within each type of the component. Using the evaluation measures of MMRE and $\text{pred}(0.25)$, it was observed that for linear relationships, small improvements obtained by GP in comparison with multiple linear regression came at the expense of the simplicity of the equations. However, it was also observed that the majority of the linear equations were rediscovered by GP. Also GP and neural networks (NN) showed superiority over multiple linear regression in case of non-linear relationship between the independent variables. The conclusion with respect to GP was that it provided similar or better values than regression equations and the GP solutions were also found to be transparent. Regolin et al. [279] used a similar approach of estimating LoC from function points and the number of components (NoC) metric. Using GP and NN, the prediction models using function points did not satisfy the criteria MMRE ≤ 0.25 and pred$(0.25) \geq 0.75$. However, the prediction models for estimating lines of code from the NoC metric were acceptable from both the NN and the GP point of view.

In [88], genetic programming and different types of standard regression analysis (linear, logarithmic, inverse quadratic, cubic, power, exponential, growth and logistic) were used to find a relationship between software size (independent variable) and cost (dependent variable). The predictive accuracy measures of pred(0.25) and MMRE showed that linear regression consistently obtained the best predictive values, with GP achieving a significant improvement over classical regression in 2 out of 12 data sets. GP performed well, pred(0.25), on most of the data sets but sometimes at the expense of MMRE. This also indicated the potential existence of over-fitting in GP solutions. It was also found that size alone as an independent variable for predicting software cost is not enough since it did not define the types of economies of scale or marginal return with clarity.

The study by Burgess et al. [56] extends the previous study from [88] by using nine independent variables to predict the dependent variable of effort measured in person hours. Using the Desharnais [85] data set of 81 software projects, the study showed that GP is consistently more accurate for MMRE but not for adjusted mean square error (AMSE), pred(0.25) and balanced mean magnitude of relative error (BMMRE). The study concluded that while GP and NN can provide better accuracy, they required more effort in setting up and training.

In [288] the authors used grammar-guided GP on 423 projects from release 7 of the ISBSG (The International Software Benchmarking Standards Group Limited [143]) data set to predict software project effort. The evaluation measures used were R-squared, MSE, MMRE, pred(0.25) and pred(0.5). In comparison with linear and log-log regression, the study showed that GP was far more accurate than simple linear regression. With respect to MMRE, log-log regression was better than GP which led to the conclusion that GP maximizes one evaluation criterion at the expense of the other. The study showed that grammar-guided GP provides both a way to represent syntactical constraints on the solutions and a mechanism to incorporate domain knowledge to guide the search process.

Lefley and Shepperd [208] used several independent variables from 407 cases to predict the total project effort comparing GP, ANN, least squares regression, nearest neighbor and random selection of project effort. With respect to the accuracy of the predictions, GP achieved the best level of accuracy the most often, although GP was found hard to configure and the resulting models could be more complex.

Tables 2.5 and 2.6[2] present the relevant summary data extracted to answer the research question from each of the primary studies within the area of software CES estimation.

2.3.3 Software fault prediction and reliability growth

Apart from studies on software quality classification (Section 2.3.1), where the program components are classified as being either *fp* or *nfp*, there are studies which are concerned with prediction of either the fault content or software reliability growth.

In [160] the authors proposed the incorporation of existing equations as a way to include domain knowledge for improving the standard GP algorithm for software fault prediction. They specifically used Akiyama's equations [15], Halstead's equation [125], Lipow's equation [213], Gaffney's equation [106] and Compton's equation [71] to add domain knowledge to a simple GP algorithm which is based on mathematical operators. Using the fitness function $(1 - \text{standard error})$, six experiments were performed using a NASA data set of 379 C functions. Five of these experiments compared standard GP with GP enhanced with Akiyama's, Halstead's, Lipow's, Gaffney's and Compton's equations. The last experiment compared standard GP with GP enhanced with all these equations simultaneously. The results showed, not surprisingly, that by including explicit knowledge in the GP solutions, the fitness values for the GP solutions increased.

In another study, [161], the same authors used another approach called data equalization to compensate for data skewness. Specifically, duplicates of interesting training instances (in this case functions with greater than zero faults) were added to the training set until the total reached the frequency of most occurring instance (in this case functions with zero faults). The fitness function used was: $1 + e^{(7*(1-n-k)/(n-1)*Se^2/Sy^2)}$, where k = number of inputs, n = number of valid results, Se = standard error and Sy = standard deviation. Using the same data sets as before, the experimental results showed that the average fitness values for the equalized data set were better than for the original data set.

In [322], grammar-guided GP was used on NASA's data set consisting of four projects to measure the probability of detection, PD (the ratio of faulty components found to all known faulty components) and false alarm rate, PF (the ratio of number of non-faulty components misclassified as faulty to all known non-faulty components).

[2] The data sets in Table 2.6 are taken at a coarser level, e.g., ISBSG data ([143]) of multiple projects is one data set.

Table 2.5: Summary data for primary studies on GP application for software CES estimation. (~) indicates indifferent results.

Fitness function	Comparison group	Evaluation measures	GP better?	Dependent variable	Article
Mean squared error	Neural networks & linear regression	pred(0.25) [a] and MMRE [b]	~	Software effort	[89]
Mean squared error	Neural networks & multiple linear regression	pred(0.25) and MMRE	~	Software size	[87]
MMRE	Neural networks	pred(0.25) and MMRE	~	Software size	[279]
Mean square error	Linear, logarithmic, inverse quadratic, cubic, power, exponential, growth and logistic regression	pred(0.25) and MMRE	~	Software cost	[88]
MMRE	neural networks	MMRE, AMSE [c], pred(0.25), BMMRE [d]	~	Software effort	[56]
Mean square error	Linear regression, log-log regression	R-squared [e], MMRE, pred(0.25) and pred(0.5)	~	Software effort	[288]
?	ANN, least squares regression, nearest neighbor and random selection of project effort	Pearson correlation coefficient of actual and predicted, AMSE, pred(0.25), MMRE, BMMRE, worst case error, the ease of set up and the explanatory value	~	Software effort	[208]

[a] Prediction at level 0.25

[b] Mean Magnitude of Relative Error

[c] Adjusted Mean Square Error

[d] Balanced Mean Magnitude of Relative Error

[e] Coefficient of multiple determination

Table 2.6: Data set characteristics for primary studies on GP application for software CES estimation.

Article	Data sets no.	Sampling of training and testing sets	Industrial (I) or academic (A)	Data sets public or private
[89]	5	a) Train and test a model with all the points. b) Train a model on 66% of the data points and test on 34% of the points	I	Public & Private
[87]	6	Train a model on 60 to 67 % of the data points and test in 40 to 37%	A	Public
[279]	2	Train on $\frac{2}{3}$ and test on $\frac{1}{3}$	I & A	Public
[88]	12	Training and testing on all data points	I & A	Public
[56]	1	Training on 63 projects, testing on 18 projects	I	Public
[288]	1	Random division of 50% in training set and 50% in testing set	I	Public
[208]	1	149 projects in the training set and 15 projects in the testing set	I	Public

The fitness function represented the coverage of knowledge represented in the individual, and equaled $\frac{tp}{(tp+fn)} * \frac{tn}{(tn+fp)}$ where *fp* is the number of false positives, tp the number of true positives, tn the number of true negatives and fn the number of false negatives. The study showed that grammar-guided GP performed better than naïve Bayes on both measures (PD and PF) in two of the projects' data while in the rest of the two data, it was better in one of the two measures.

We were also able to find a series of studies where the comparison group included traditional software reliability growth models. Zhang et al. [355] used mean time between failures (MTBF) time series to model software reliability growth using genetic programming, neural networks (NN) and traditional software reliability models, i.e., Schick-Wolverton, Goel-Okumoto, Jelinki-Moranda and Moranda. Using multiple evaluation measures of short-term range error, prequential likelihood, model bias, model bias trend, goodness of fit and model noise; the GP approach was found better than the traditional software reliability growth models. However, it is not clear from the study how neural networks performed against all the evaluation measures (except for the short-term range error where GP was better than neural networks). Also it is not clear from the study what sampling strategy was used to split the data set into training and testing set. The fitness function information is also lacking from the study. The study is however extended in [356] with adaptive cross-over and mutation probabilities, and the corresponding GP was named adaptive genetic programming. In comparison with standard GP and the same reliability growth models (as used in the previous study), the mean time between failures (MTBF) and the next mean time to failure (MTTF) values for adaptive GP were found to be more accurate.

Afzal and Torkar [6] used fault data from three industrial software projects to predict the software reliability in terms of number of faults. Three SRGMs (Goel-Okumoto, Brooks and Motley, and Yamada's *S*-shaped) were chosen for comparison using the fitness function of sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$, where e_i is the actual fault count data,

$e_i^{'}$ the estimated value of the fault count data and n the size of the data set used to train the GP models. The faults were aggregated on weekly basis and the sampling strategy divided the first $\frac{2}{3}$ of the data set into a training set and remaining $\frac{1}{3}$ into a test set. Using prequential likelihood ratio, adjusted mean square error (AMSE), Braun statistic, Kolmogorov-Smirnov tests and distribution of residuals, the GP models were found to be more accurate for prequential likelihood ratio and Braun statistic but not for AMSE. The goodness of fit of the GP models were found to be either equivalent or better than the competing models used in the study. The inspection of the models' residuals also favored GP.

In [73], the authors used GP and GP with boosting to model software reliability. The comparisons were done with time based reliability growth models (Jelinski-Moranda and geometric), coverage-based reliability growth model (coverage-based binomial model) and artificial neural network (ANN). The evaluation measures used for time-based models included maximum deviation, average bias, average error, prediction error and correlation coefficient. For coverage-based models, an additional Kolmogorov-Smirnov test was also used. The fitness function used was weighted root mean square error (WRMSE), $\sqrt{\sum_{i=1}^{m}\left(x_i - x_i^d\right)^2 D_i m}$ where x_i = real value, x_i^d = estimated value, D_i = weight of each example and m = size of the data set. Using the first $\frac{2}{3}$ of the data set as a training set, it was observed that GP with boosting (GPB) performed better than traditional models for models based on time. However, there was no statistical difference between GP, GPB and ANN models. For models based on test coverage, the GPB models' results were found to be significantly better compared to that of the GP and ANN models.

In [262], the authors used a modified GP algorithm called the $\mu + \lambda$ GP algorithm to model software reliability growth. In the modified algorithm, $n\%$ of the best individuals were applied the genetic operators in each generation. The genetic operators generated λ individuals, which competed with their parents in the selection of μ best individuals to the next generation where ($\lambda > \mu$). The fitness function used was root mean square error (RMSE), given by: $\sqrt{\frac{\sum_{i=1}^{n}|x_i - x_i^d|}{n}}$ where x_i is the real value, x_i^d is the estimated value and n is the size of the data set. Using measures as maximum deviation, average bias, average error, prediction error and correlation coefficient, the results favored modified GP algorithm. Additional paired two-sided t-tests for average error confirmed the results in favor of modified GP with a statistically significant difference in the majority of the results between the modified and standard GP algorithm.

Table 2.7 and Table 2.8[3] shows the relevant summary data extracted to answer the

[3] The data sets in Table 2.8 are taken at a coarser level, e.g., ISBSG data ([143]) of multiple projects is one data set.

research question from each of the primary studies within the area of software fault prediction and reliability growth.

2.4 Discussion and areas of future research

Our research question was initially posed to assess the efficacy of using GP for prediction and estimation in comparison with other approaches. Based on our investigation, this research question is answered depending upon the prediction and estimation of the attribute under question. In this case, the attribute belonged to three categories:

1. Software fault proneness (software quality classification).
2. Software CES estimation.
3. Software fault prediction and software reliability growth modeling.

For software quality classification, seven out of nine studies reported results in favor of using GP for the classification task. Two studies were inconclusive in favoring a particular technique either because the different measures did not converge, as in [280], or the proposed technique used GP for initial investigative purposes only, without being definitive in the judgement of GP's efficacy, as in [278] (these two studies are indicated by the sign ~ in Table 2.3).

The other seven studies were co-authored by similar authors to a large extent and the data sets also overlapped between studies but these studies contributed by introducing different variations of the GP fitness function and also used different comparison groups. These seven studies were in agreement that GP is an effective method for software quality classification based on comparisons with neural networks, *k*-nearest neighbor, linear regression and logistic regression. Also GP was used to successfully rank-order software components in a better way than the random ranking and the ranking done on the basis of lines of code. Also it was shown that numerous enhancements to the GP algorithm are possible hence improving the evolutionary search in comparison with standard GP algorithm. These enhancements include random subset selection and different mechanisms to control excessive code growth during GP evolution. Improvements to the GP algorithm gave better results in comparison with standard GP algorithm for two studies [180, 216]. However, one finds that there can be two areas of improvement in these studies: (*i*) Increasing the comparisons with more techniques. (*ii*) Increasing the use of public data sets.

Table 2.7: Summary data for primary studies on GP application for software fault prediction and reliability growth. (?) indicates absence of information and (~) indicates indifferent results.

Fitness function	Comparison group	Evaluation measures	GP better?	Dependent variable	Article
$1 - \text{standard error}$	Standard GP	Fitness values	√	SW fault prediction	[160]
$1 + e^{(7*(1-n-k)/(n-1)*Se^2/Sy^2)}$	Standard GP	Fitness values	√	SW fault prediction	[161]
$\frac{tp}{(tp+fn)} * \frac{tn}{(tn+fp)}$	Naive Bayes	PD & PF	~	SW fault prediction	[322]
?	Neural networks and traditional software reliability growth models	Short-term range error, prequential likelihood, model bias, model bias trend, goodness of fit and model noise	√	Software reliability	[355]
?	Traditional software reliability growth models	Mean time between failures and next mean time to failure	√	Software reliability	[356]
$\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$	Traditional software reliability growth models	Prequential likelihood ratio, AMSE, Braun statistic, Kolmogorov-Smirnov test and distribution of residuals	√	Software reliability	[6]
WRMSE [a]	Traditional software reliability growth models and ANN	Maximum deviation, average bias, average error, prediction error, correlation coefficient, Kolmogorov-Smirnov	√	Software reliability	[73]
RMSE [b]	Standard GP	Maximum deviation, average bias, average error, prediction error and correlation coefficient	√	Software reliability	[262]

[9] Weighted root mean square error
[10] Root mean square error

[a] Weighted root mean square error
[b] Root mean square error

Table 2.8: Data set characteristics for primary studies on GP application for software fault prediction and reliability growth. (?) indicates absence of information.

Article	Data sets no.	Sampling of training and testing sets	Industrial (I) or academic (A)	Data sets public or private
[160]	1	?	I	Public
[161]	1	?	I	Public
[322]	1	10-fold cross-validation	I	Public
[355]	1	?	I	Private
[356]	1	?	I	Private
[6]	3	First $\frac{2}{3}$ of the data set for training and the rest for testing	I	Private
[73]	2	First $\frac{2}{3}$ of the data set for training and the rest for testing	I	Public & Private
[262]	1	First $\frac{2}{3}$ of the data set for training and the rest for testing	I	Public

We also observe from Table 2.3 that multi-objective GP is an effective way to seek near-optimal solutions for software quality classification in the presence of competing constraints. This indicates that further problem-dependent objectives can possibly be represented in the definition of the fitness function, which potentially can give better results. We believe that in order to generalize the use of GP for software quality classification, the comparison groups need to increase in size.

There are many different techniques that have been applied by researchers to software quality classification, see e.g., [209]. GP needs to be compared with a more representative set of techniques that have been found successful in earlier research—only then can we be able to ascertain that GP is a competitive technique for software quality classification. We see from Table 2.4 that all the data sets were private. In this regards, the publication of private data sets needs to be encouraged. Publication of data sets would encourage other researchers to replicate the studies based on similar data sets and, hence, we can have greater confidence in the correctness of the results. Nevertheless, one encouraging trend that is observable from Table 2.4 is that the data sets represented real world projects which adds to the external validity of these results.

For software CES estimation, there was no strong evidence of GP performing consistently on all the evaluation measures used (as shown in Table 2.5). The sign ~ in the last column of Table 2.5 shows that the results are inconclusive concerning GP. The study results indicate that while GP scores higher on one evaluation measure, it lags behind on others.

There is also a trade-off between different qualitative factors, e.g., complexity of interpreting the end solution, and the ease of configuration and flexibility to cater for varying data sets. The impression from these studies is that GP also requires some effort in configuration and training. There can be different reasons related to the experimental design for the inconsistent results across the studies using GP for software CES estimation. One reason is that the accuracy measures used for evaluation purposes are not near to a standardized use. While the use of pred(0.25) and MMRE are commonly used, other measures, including AMSE and BMMRE, are also applied. It is important that researchers are aware of the merits/demerits of using these evaluation measures [105, 289]. Another aspect which differed between the studies was the sampling strategies used for training and testing sets (Column 3, Table 2.6). These different sampling strategies are also a potential contributing factor in inconsistent model results. What is also observable from these studies is that over-fitting is a common problem for GP. However, there are different mechanisms to avoid over-fitting, such as random subset selection on the training set and placing limits on the size of the GP trees. These mechanisms should be explored further.

As previously pointed out in Section 2.3.2, Crespo et al. [76] identified six artificial intelligence techniques applicable to software development effort estimation. It is interesting to note that our literature search did not find any study, which compares all of these techniques. As for the studies related to software fault prediction and software reliability growth, seven out of eight studies favor the use of GP in comparison with neural networks, naïve Bayes and traditional software reliability growth models (this is evident from the last column in Table 2.7). However, as Table 2.8 showed, it was not clear from four studies which sampling strategies were used for the training and testing sets. From two of these four studies, it was also not clear what fitness function was used for the GP algorithm. If, however, we exclude these four studies from our analysis, GP is still a favorable approach for three out of four studies. With respect to comparisons with traditional software reliability growth models, the main advantage of GP is that it is not dependent on the assumptions that are common in these software reliability growth models. Also GP promises to be a valid tool in situations where different traditional models have inconsistent results. Besides, we also observe that several improvements to the standard GP algorithm are possible which provides comparatively better results. Specifically, we see studies where the incorporation of explicit domain knowledge in the GP modeling process has resulted in improved fitness values [160]. Table 2.7 also shows that the variety of comparison groups is represented poorly; there is an opportunity to increase the comparisons with more techniques and also to use a commonly used technique as a baseline.

For studies which were inconclusive in the use of GP for prediction/estimation, we include quotations from the respective papers in Table 2.9 (an approach similar to the one used in [228]) that reflects the indifference between GP and other approaches.

What is evident from these studies is the following:

Table 2.9: Summary of the studies showing inconclusive results in using GP.

Quotes	Article
While generally not as good as the results obtained from other methods, the GP results are reasonably accurate but low on coverage.	[280]
The rate of successful classifications for training data is around 66 and 72% for the first architect and the second architect, respectively. In the case of testing data the rates are 55 and 63%.	[278]
However, from the point of view of making good predictions, no technique has been proved to be clearly superior. …From the values shown in the tables, there is no great superiority of one method versus the others …GP can be used as an alternative to linear regression, or as a complement to it.	[89]
The final impression is that GP has worked very well with the data used in this study. The equations have provided similar or better values than the regression equations. Furthermore, the equations are "intelligible", providing confidence in the results. …In the case of linear relationships, some of the small improvements obtained by GP compared to MLR come at the expense of the simplicity of the equations, but the majority of the linear equations are rediscovered by GP.	[87]
We cannot conclude GP is a better technique than NN. …GP and ANN are valid and promising approaches to size estimation. …However, GP presents additional advantages with respect to NN. The main advantage of using GP is the easy interpretation of result.	[279]
From the point of view of the capabilities of the two methods, GP achieves better values in the pred(0.25) in eleven out of the twelve data sets, but sometimes at the cost of having a slight worse value of the MMRE. Only in data sets A and H, GP provides a significant improvement over classical regression.	[88]
There is evidence that GP can offer significant improvements in accuracy but this depends on the measure and interpretation of accuracy used. GP has the potential to be a valid additional tool for software effort estimation but set up and running effort is high and interpretation difficult …	[56]
Log regression models perform much worse than GP on MSE, about the same as GP on R^2 and pred(0.25), and better than GP on MMRE and pred(0.5). One way of viewing this is that GP has more effectively fit the objective, namely minimizing MSE, at the cost of increased error on other measures.	[288]
The results do not find a clear winner but, for this data, GP performs consistently well, but is harder to configure and produces more complex models.	[208]
In two of the databases, our model is proved superior to the existing literature in both comparison variables, and in the rest two databases, the system is shown better in one of the two variables.	[322]

1. The accuracy of GP as a modeling approach is attached to the evaluation measure used. The impression from these studies is that GP performs superior on one evaluation measure at the cost of the other. This indicates that the GP fitness function should not only be dependent on the minimization of standard error but also biased in searching those solutions which reflect properties of other evaluation measures, such as correlation coefficient.

2. The qualitative scores for GP models are both good and bad. While they might be harder to configure and result in complex solutions, the results can nevertheless be interpreted to some extent. This interpretation can be in the form of identifying the few significant variables [198]. But another key question is that whether or not we are able to have a reasonable explanation of the relationship between the variables. As an example, Dolado [87] provides the following equation generated by GP:

   $LoC = 50.7 + 1.501 * \mathit{data\ elements} + \mathit{data\ elements} * \mathit{relations} - 0.5581 * \mathit{relations}$

   While this equation identifies the dependent variables, it is still difficult to *explain* the relationships. Simplification of resulting GP solutions is thus important.

Based on the above discussion, we can conclude that while the use of GP as a prediction tool has advantages, as presented in Section 2.3, there are, at the same time, challenges to overcome as points 1 and 2 indicate above. We believe that these challenges offer promising future work to undertake for research community.

2.5 Empirical validity evaluation

We assume that our review is based on studies which were unbiased. If this is not the case, then the validity of this study is also expected to suffer [228]. Also, like any other systematic review, this one too is limited to making use of information given in the primary studies [191]. There is also a threat that we might have missed a relevant study but we are confident that both automated and manual searches of the key information sources (Section 2.2.2) have given us a complete set. Our study selection procedure (Section 2.2.3) is straightforward and the researchers had agreement on which studies to include/exclude. However, this review does not cover unpublished research that had undesired outcome and company confidential results.

2.6 Conclusions

This systematic review investigated whether symbolic regression using genetic programming is an effective approach in comparison with machine learning, regression techniques and other competing methods. The results of this review resulted in a total of 24 primary studies; which were further classified into software quality classification (nine studies), software CES estimation (seven studies) and fault prediction/software reliability growth (eight studies).

Within software quality classification, we found that in seven out of nine studies, GP performed better than competing techniques (i.e., neural networks, *k*-nearest neighbor, linear regression and logistic regression). Different enhancements to the standard GP algorithm also resulted in more accurate quality classification, while GP was also more successful in rank-ordering of software components in comparison with random ranking and ranking based on lines of code. We concluded that GP seems to be an effective method for software quality classification. This is irrespective of the fact that one author was part of seven out of nine primary studies and the fact that there was an overlap of data sets used across the studies. We considered each of these primary studies representing a distinct contribution in terms of different algorithmic variations.

For software CES estimation, the study results were inconclusive in the use of GP as an effective approach. The main reason being that GP optimizes one accuracy measure while degrades others. Also the experimental procedures among studies varied, with different strategies used for sampling the training and testing sets. We were therefore inconclusive in judging whether or not GP is an effective technique for software CES estimation.

The results for software fault prediction and software reliability growth modeling leaned towards the use of GP, with seven out of eight studies resulting in GP performing better than neural networks, naïve Bayes and traditional software reliability growth models. Although four out of these eight studies lacked in some of the quality instruments used in Table 1 (Appendix A, page 281); still three out of the remaining four studies reported results in support of GP. We therefore conclude that current literature provides evidence in support of GP being an effective technique for software fault prediction and software reliability growth modeling.

Based on the results of the primary studies, we can offer the following recommendations. Some of these recommendations refer to other researchers' guidelines which are useful to reiterate in the context of this study:

1. Use public data sets whenever possible. In case of private data sets, there are ways to transform the data sets to make it public domain (e.g., one such transformation is discussed in [343]).

2. Apply commonly used sampling strategies to help other researchers replicate, improve or refute the established predictions and estimations. From our sample of primary studies, the sampling strategy of $\frac{2}{3}$ for training, remaining $\frac{1}{3}$ for testing and 10-fold cross validation are mostly used. Kitchenham et al. [191] recommends using a jackknife approach with leave-one-out cross-validation process; this needs to be validated further.

3. Avoiding over-fitting in GP solutions is possible and is beneficial to increase the generalizability of model results in the testing data set. The primary studies in this review offer important results in this regards.

4. Always report the settings used for the algorithmic parameters (also suggested in [30]).

5. Compare the performances against a comparison group which is both commonly used and currently an active field of research. For our set of primary studies, comparisons against different forms of statistical regression and artificial neural networks can be seen as a baseline for comparisons.

6. Use statistical experimental design techniques to minimize the threat of differences being caused by chance alone [249].

7. Report the results even if there is no statistical difference between the quality of the solutions produced by different methods [30].

The next Chapter 3 discusses the mechanism enabling GP to progressively search for better solutions, explores the use of GP for software fault count predictions and compares the predictive capabilities of GP with three traditional software reliability growth models.

Chapter 3

Genetic programming for software fault count predictions

Originally published in: Proceedings of the 1st International Conference on Computer Science and its Applications (CSA'08), Proceedings of the 12th IEEE International Multitopic Conference (INMIC'08) and Proceedings of the 3rd International Conference on Software Engineering Advances (ICSEA'08)

W. Afzal, R. Torkar and R. Feldt

3.1 Introduction

Software has become a key element in the daily life of individuals and societies as a whole. We are increasingly dependent on software and because of this ever-increasing dependency; software failures can lead to hazardous circumstances. Ensuring that the software is of high quality is thus a high priority. A key element of software quality is software reliability, defined as the ability of a system or component to perform its required functions under stated conditions for a specific period of time [301]. If the software frequently fails to perform according to user-specified behavior, other software quality factors matters less [246]. It is, therefore, imperative that the reliability of the software is determined before making it operational.

Deciding upon when to release the software is also important because releasing software that contains faults will result in high failure costs whereas, on the other hand, prolonged debugging and testing increases development costs. Reliability growth modeling is an important criterion, which helps in making an informed decision about when to release the software. A software reliability growth model (SRGM) describes the mathematical relationship of finding and removing faults to improve software reliability. A SRGM performs curve fitting of observed failure data by a pre-specified model formula, where the parameters of the model are found by statistical techniques like e.g., the maximum likelihood method [251]. The model then estimates reliability or predicts future reliability by different forms of extrapolation [224]. After the first software reliability growth model was proposed by Jelinski and Moranda in 1972 [146], there have been numerous reliability growth models following it. These models come under different classes [223], e.g., exponential failure time class of models, Weibull and Gamma failure time class of models, infinite failure category models and Bayesian models. The existence of a large number of models requires a user to select and apply an appropriate model. For practitioners, this may be an unmanageable selection problem and there is a risk that the selected model is unsuitable to the particulars of the project in question.

Some models are complex with many parameters. Without extensive mathematical background, practitioners cannot determine when a model is applicable and when the model diverges from reality. Even if the dynamics of the testing process are well known, there is no guarantee that the model whose assumptions appear to best suit these dynamics will be most appropriate [257]. Moreover, these *parametric* software reliability growth models are often characterized by a number of assumptions, e.g., that once a failure occurs the fault that caused the failure is immediately removed and that the fault removal process will not introduce new faults. These assumptions are often unrealistic (see e.g., [344]), hence, causing problems in the long-term applicability and validity of these models. Under these constraints, what becomes significantly interesting is to have modeling mechanisms that can exclude the pre-suppositions about the model and are based entirely on the fault data. In this respect, genetic programming (GP) can be used as an effective tool because, being a *non-parametric* method, GP does not conceive a particular structure for the resulting model and GP also does not make any assumptions about the distribution of the data.

This chapter presents a multi-stage exploration of using GP for the purpose of predicting software reliability. Stage one discusses the mechanisms enabling GP to potentially be an effective modeling technique. Stage two presents an experiment where we apply GP to evolve a model based on weekly fault count data. The contribution of this stage is exploring the use of GP as a potential method for software fault count predictions. We use five different measures to evaluate the adaptability and predictive ability of the GP-evolved model on three sets of fault data that corresponds to three projects carried out by a large telecommunication company. The results of the experiment indicate that software reliability growth modeling is a suitable problem domain for GP as the GP evolved model gives statistically significant results for goodness of fit and predictive accuracy on each of the data sets. Stage three presents the results of the comparison between models evolved using GP and three other traditional SRGMs based on the same data sets as in stage two. Stage three compares the models using measures of model validity, goodness of fit and residual analysis. The comparative results indicate that in terms of model validity, two out of three measures favor GP evolved models. The GP evolved models also represented comparatively better goodness of fit, while residual analysis showed that the predictions from the GP evolved model are comparatively less biased.

The remainder of this chapter is organized as follows. Section 3.2 and Section 3.3 present related work and a background to genetic programming, respectively. Section 3.4 discusses stage one of the study. The second stage is discussed in Section 3.5 and consists of a discussion on the research method, experimental setup, results and summary of results. The third stage is discussed in Section 3.6, consisting of a discussion about selection of traditional SRGMs, hypotheses, evaluation measures, results and a summary of results. The validity evaluation of the complete study appears in Section 3.7, while the chapter ends with a discussion and conclusions in Section 3.8 and Section 3.9, respectively.

3.2 Related work

Within the realm of machine learning algorithms, there has been work exploring the use of artificial neural networks for software reliability growth modeling (e.g., [293]), but our focus here is on the research done using GP for software reliability growth modeling.

Studies reporting the use of GP for software reliability modeling are few and recent. Costa et al. [75] presented the results of two experiments exploring GP models based on time and test coverage. The authors compared the results with other traditional and non-parametric artificial neural network (ANN) models. For the first experiment, the authors used 16 data sets containing time-between-failure (TBF) data from projects related to different applications. The models were evaluated using five different measures, four of these measures represented different variants of differences between observed and estimated values. The results from the first experiment, which explored GP models based on time, showed that GP adjusts better to the reliability growth curve. Also GP and ANN models converged better than traditional reliability growth models. GP models also showed lowest average error in 13 out of 16 data sets. For the second experiment, which was based on test coverage data, a single data set was used. This time the Kolmogorov-Smirnov test was also used for model evaluation. The results from the second experiment showed that all measurements were consistently better for GP and ANN models. The authors later extended GP with boosting techniques for reliability growth modeling [262] and reported improved results.

A similar study by Zhang and Chen [355] used GP to establish a software reliability model based on mean time between failures (MTBF) time series. The study used a single data series and used six different criteria for evaluating the GP evolved model. The results of the study also confirmed that in comparison with the ANN model and traditional models, the model evolved by GP had higher prediction precision and better applicability.

There are several ways in which the present work differs from the aforementioned studies. Firstly, none of the previous studies used data sets consisting of weekly fault count data. In this study, our aim is to use the weekly fault count data as a means to evolve the reliability growth model using GP. Secondly, we have avoided performing any pre-processing of data to avoid chances of incorporating bias. Thirdly, in our study, we remain consistent throughout with using 2/3 of the data to build the model and use the rest 1/3 of the data for model evaluation for all of our data sets. This splitting procedure was found not to be consistent in earlier studies. Lastly, we do not change the evaluation measures for all the data sets, in an attempt to provide a fair evaluation. This is again something that is lacking from earlier studies.

3.3 Background to genetic programming

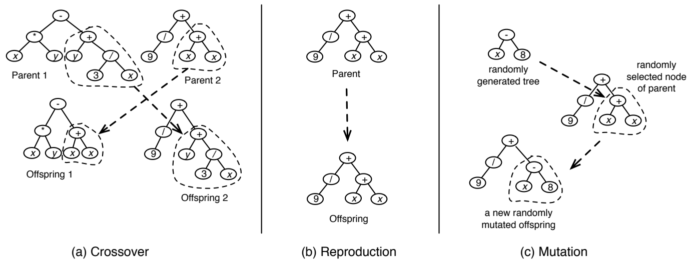
The evolution of software reliability growth models using GP is an example of a symbolic regression problem. Symbolic regression is an error-driven evolution as it aims to find a function, in symbolic form, that fits (or approximately fits) data from an unknown curve [200]. In simpler terms, symbolic regression finds a function whose output matches some target values. GP is well suited for symbolic regression problems, as it does not make any assumptions about the structure of the function.

GP is an evolutionary computation technique (first results reported by Smith [295] in 1980) and is an extension of genetic algorithms. As compared with genetic algorithms, the population structures (individuals) in GP are not fixed length character strings, but programs that, when executed, are the candidate solutions to the problem.

Poli et al. [272] define GP as:

> “GP is a systematic, domain-independent method for getting computer to solve problems automatically starting from a high-level statement of what needs to be done.”

Programs are expressed in GP as syntax trees, with the nodes indicating the instructions to execute and are called functions (e.g., *min*, $*$, $+$, $/$), while the tree leaves are called terminals which may consist of independent variables of the problem and random constants (e.g., *x*, *y*, 3). The fitness evaluation of a particular individual is determined by the correctness of the logical output produced for all of the fitness cases [27]. The fitness function guides the search in promising areas of the search space and is a way of communicating a problem’s requirements to the GP algorithm. The control parameters limit and control how the search is performed like setting the population size and probabilities of performing the genetic operations. The termination criterion specifies the ending condition for the GP run and typically includes a maximum number of generations [57]. GP iteratively transforms a population of computer programs into a new generation of programs using various genetic operators. Typical operators include crossover, mutation and reproduction. It is expected that over successive iterations, more and more useful structures or programs be evolved, eventually resulting in a structure having most useful sub-components. That structure would then represent the optimal or near-optimal solution to the problem. The crossover operator creates new structure(s) by combining randomly chosen parts from two selected programs or structures (Figure 3.1a). The mutation operator creates a new structure by randomly altering a chosen part of a program (Figure 3.1b). The reproduction operator simply copies a selected structure to the new population (Figure 3.1c). Figure 3.2 shows the



(a) Crossover

(b) Reproduction

(c) Mutation

Figure 3.1: Crossover, reproduction and mutation operators in GP.

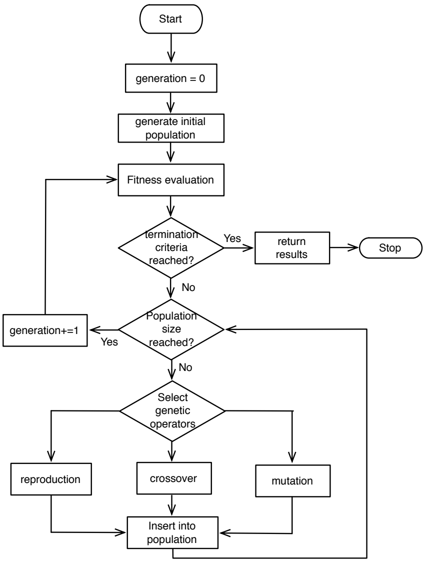


Figure 3.2: The GP process.

flowchart of the GP process.

3.4 Study stage 1: GP mechanism

The suitability of GP for modeling software reliability growth is based on the identification of building blocks and progressively improving overall fitness.

According to Koza [200], the GP population contains building blocks, which could be any GP tree or sub-tree in the population, and according to the building block hypothesis, good building blocks improve the fitness of individuals that include them, and these individuals have greater chance to be selected for reproduction. Therefore, good building blocks get combined to form better individuals [29]. This hypothesis appears suited to adaptive model-building that can be used for predicting software reliability growth.

The evolution of better individuals using GP is shown in Figure 3.3.

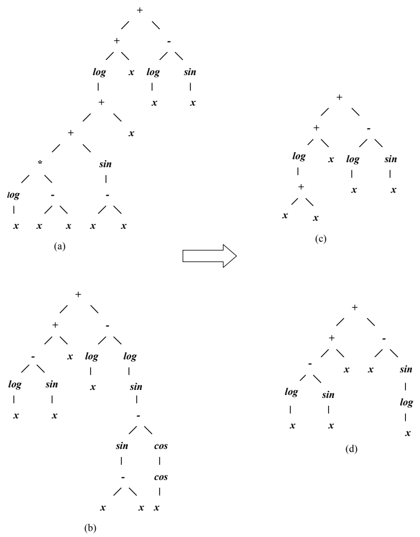


Figure 3.3: Combination of trees containing building blocks.

The fitness of a GP solution is the sum of absolute differences between the obtained and expected results in all fitness cases. Suppose that during the fourth generation of a GP run, two solutions have evolved (see Figure 3.3a and 3.3b in Figure 3.3) containing different building blocks for an optimum solution. For tree 1 (Figure 3.3a), the sum of absolute differences between the obtained and expected results in all fitness cases was 31.34, while for tree 2 (Figure 3.3b), the fitness measure was 28.9. By combining these two trees, two new trees could emerge (Figure 3.3c and Figure 3.3d). The first tree (Figure 3.3c) has a better fitness of 27.8 than any of its parents, while the second tree (Figure 3.3d) produced a higher fitness of 39.

In order to evolve a general function based on the fitness cases, the search space of solutions can get complex. This increase in complexity helps the GP programs comply with all the fitness cases [272]. Evolutionary algorithms have been found to be robust for complex search spaces and genetic programming can potentially be a valid technique to evolve software reliability growth models because the suitability of genetic programming has already been proven for symbolic regression and curve fitting problems. Being a stochastic search technique, the different runs of GP would result in different trajectories [272]. Figure 3.4 shows how the GP algorithm is searching the program space of solutions to track the model to approximate.

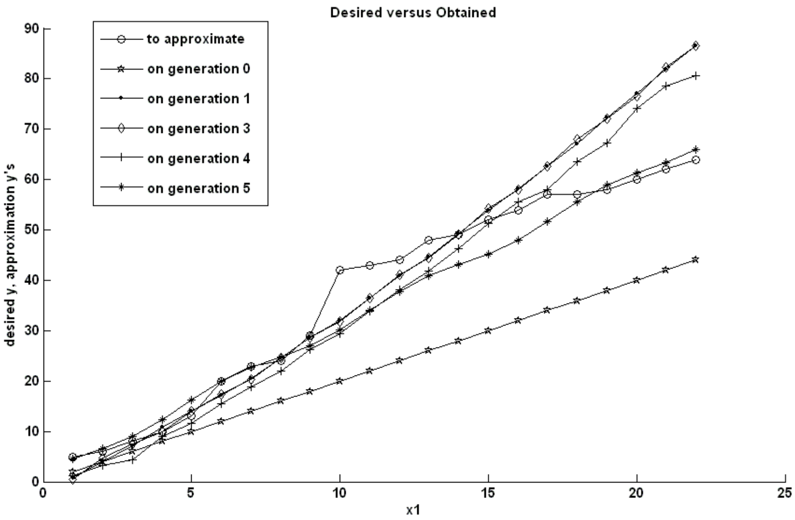


Figure 3.4: Several approximations to the original fault count data in different generations.

Figure 3.5 shows the Pareto front when modeling software reliability growth for one of the data sets. A Pareto front consists of a set of Pareto optimal solutions. A Pareto optimal solution is a non-dominated solution since it is not dominated by any other feasible solution in the entire search space [237]. The Pareto front in Figure 3.5 shows the set of solutions for which no other solution was found which both had a smaller tree and better fitness [292]. The Figure 3.5 also shows the best fitness found for each tree size. It is clear from Figure 3.5 that the fitness of different solutions fluctuates as the number of nodes increases during the course of generations.

3.5 Study stage 2: Evaluation of the predictive accuracy and goodness of fit

In this stage, we present the details of an experiment where we use GP as a potential method for software fault count predictions.

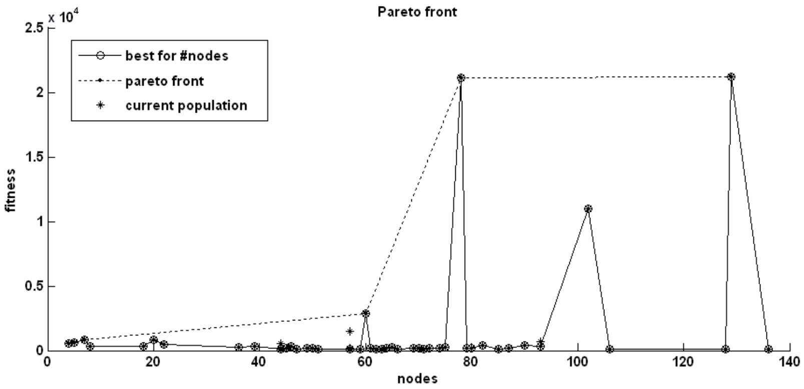


Figure 3.5: Visualization of Pareto front for one set of industrial fault count data.

3.5.1 Research method

The discussion regarding the research method includes a description of the data sets used, the formulated hypotheses and a description of the evaluation measures.

Fault count data sets

The data sets used in this study are based on the weekly fault count data collected during the testing of three large-scale software projects at a large telecom company. The projects are targeted towards releases of three mature systems that have been on the market for several years. These projects follow an iterative development process meaning that within each iteration, a new system version, containing new functionality and fixes of previously discovered faults, is delivered to test. These iterations occur on a weekly basis or even more frequently, while testing of new releases proceed continuously. In this scenario, it becomes important for project managers to estimate the current reliability and to predict the reliability ahead of time, so as to measure the quality impact with continuous addition of new functionality and fixes of previously discovered faults. The three projects are similar in size, i.e., they have approximately half a million lines of code. There are, however, minor differences with respect to the projects' duration. The first project lasted 26 weeks, whereas the second and third projects lasted 33 and 30 weeks respectively.

The independent variable in our case was the week number while the corresponding dependent variable was the count of faults. We used $2/3$ of the data in each data set for building the model and $1/3$ of the data for evaluating the model according to the five different measures (Subsection 3.5.1). This implies that we are able to make predictions on several weeks constituting $1/3$ of the data.

Hypothesis

The purpose of this experiment is to evaluate the predictive accuracy and goodness of fit of GP in modeling software reliability using weekly fault count data collected in an industrial context. In order to formalize the purpose of the experiment, we define the following hypotheses:

$\mathrm{H}_{0-\mathrm{acc}}$: GP model does not produce significantly accurate predictions.

$\mathrm{H}_{1-\mathrm{acc}}$: GP model produces significantly accurate predictions.

$\mathrm{H}_{0-\mathrm{gof}}$: GP model does not fit significantly to a set of observations.

$\mathrm{H}_{1-\mathrm{gof}}$: GP model fits significantly to a set of observations.

In order to test the above hypotheses, we use five measures for evaluating the goodness of fit and predictive accuracy as detailed in the next section.

Evaluation measures

It is usually recommended to use more than one measure to determine model applicability, as in [257], because reliance on a single measure can lead to making incorrect choices. The deviation between observed and the fitted values were, in our case, measured using a goodness-of-fit test. We selected two measures for determining the goodness of fit: the two-sample two-sided Kolmogorov-Smirnov (K-S) test and Spearman's rank correlation coefficient. For measuring predictive accuracy, we used prediction at level l, mean magnitude of relative error (MMRE) and a measure of prediction stability. What follows is a brief description of each of these measures and how they were used in this study.

Kolmogorov-Smirnov The K-S test is a commonly used statistical test for measuring goodness of fit [231, 305]. The K-S test is a distribution-free test for measuring general differences in two populations.

The null hypothesis of interest here is that the two samples, $F(t)$ and $G(t)$ have the same probability distribution and represents the same population.

$$H_0 : [F(t) = G(t), \text{for every } t]$$

We have used the significance level $\alpha = 0.05$ and if the K-S statistic J is greater than or equal to the critical value J_α, the null hypothesis is rejected in favor of the alternate hypothesis; otherwise we conclude that the two samples have the same distribution. For detailed description of the test, see [137].

Spearman's rank correlation coefficient Spearman's rank correlation coefficient ρ is the non-parametric counterpart of the parametric linear correlation coefficient, r.

We use hypothesis testing to determine the strength of relationship between observed and estimated model values. If the absolute value of the computed value of ρ exceeds the critical values of ρ for $\alpha = 0.05$, we conclude that there is a significant relationship between the observed and estimated model values. Otherwise, there is not sufficient evidence to support the conclusion of a significant relationship between the two distributions. More details on Spearman's rank correlation coefficient can be found in [159].

Prediction at level l Prediction at level l, or $pred(l)$, represents the count of the number of predictions within $l\%$ of the actuals. We have used the standard criterion for considering a model as acceptable which is $pred(0.25) \geq 0.75$ which means that at least 75% of the estimates are within the range of 25% of the actual values [87].

Mean magnitude of relative error Mean magnitude of relative error (MMRE) is the most commonly used accuracy statistic.

Conte et al. [72] consider MMRE ≤ 0.25 as acceptable for effort prediction models; we use the same measure for our study.

Measure of prediction stability The predictions of a model should not vary significantly and should remain stable to denote the maturity of the model. We use here a good rule of thumb given in [343] for prediction stability which says that a prediction is stable if the prediction in week i is within 10% of the prediction in week $i-1$.

3.5.2 Experimental setup

In this study we used MATLAB version 7.0 [230] and GPLAB version 3.0 [292] (a GP toolbox for MATLAB).

Control parameter selection for GP

GPLAB allows for different choices of tuning control parameters. We were able to adjust the control parameters after a certain amount of experimentation. We experimented with different function sets and terminal sets by fixing the rest of the control parameters like population size, number of generations and sampling strategy. Initially we experimented with a minimal set of functions by keeping the terminal set containing the independent variable only. We incrementally increased the function set with additional functions and later on also complemented the terminal set with a random constant. For each data set, the best model having the best fitness was chosen from all the runs of the GP system with different variations of function and terminal sets. The function set for Project 1 and Project 3 data sets was the same, while a slightly different function set for Project 2 gave the best fitness. The GP programs were evaluated according to the sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$, where e_i is the actual fault count data, $e_i^{'}$ is the estimated value of the fault count data and n is the size of the data set used to train the GP models. The control parameters that were chosen for the GP system are shown in Table 3.1.

Table 3.1: Main control parameters used for the GP system.

Control Parameter	Value
Population size	30
Number of generations	200
Termination condition	200 generations
Function set (for project 1 & 3)	$\{+,-,*,sin,cos,log\}$
Function set (for project 2)	$\{+,-,*,/,sin,cos,log\}$
Terminal set	$\{x\}$
Tree initialization	ramped half-and-half
Initial maximum number of nodes	28
Maximum number of nodes after genetic operations	512
Genetic operators	crossover, mutation, reproduction
Selection method	lexictour
Elitism	replace

3.5.3 Results

In this section, we describe the results of the evaluation measurements to assess the adaptability and predictive accuracy of the GP evolved model.

Adaptability of the model

Table 3.2 shows the statistic J for the K-S test performed on the validation fault count data ($1/3$ of the original data set) and the estimated fault count data provided by the GP evolved model for each of the data sets. The critical values J_α for $\alpha = 0.05$ are also given. We selected the significance level (α) of 0.05 as it is common in practice [158]. We see that in each data set, $J < J_\alpha$; this suggests that the estimated fault count data, as provided by the GP model, fits quite well to the set of observations in all three data sets.

Table 3.2: Results of applying two-sample two-sided Kolmogorov-Smirnov test.

	J	$J_{\alpha=0.05}$	Sample size	$J < J_\alpha$
Project 1	0.40	0.70	10	$\surd$
Project 2	0.27	0.64	11	$\surd$
Project 3	0.10	0.70	10	$\surd$

We additionally calculated the Spearman's rank correlation coefficient ρ for determining the relationship between actual and estimated model values (Table 3.3). At significance level $\alpha = 0.05$, computed values of ρ exceeds the critical values $r_{\alpha=0.05}$ for every data set. This indicates that there is a strong relationship between actual values and estimated model values.

Table 3.3: Results of applying Spearman's correlation coefficient test.

	ρ	$r_{\alpha=0.05}$	Sample size	$\rho > r_{\alpha=0.05}$
Project 1	0.99	0.65	10	$\surd$
Project 2	0.93	0.62	11	$\surd$
Project 3	1.00	0.65	10	$\surd$

Based upon the results of applying Kolmogorov-Smirnov and Spearman's rank correlation coefficient, we are able to reject the null hypothesis, $\mathrm{H}_{0-\mathrm{gof}}$ in support of the alternative hypothesis, $\mathrm{H}_{1-\mathrm{gof}}$.

Measuring predictive accuracy

Table 3.4 presents the results of measuring $pred(0.25)$ for the three data sets where e_i denotes the actual fault count data and $e_i^{'}$ is the estimated value of the fault count data.

Table 3.4: Testing for $\mathbf{pred(0.25) \geq 0.75}$.

Week i	25% of e_i	$e_i^{'}$	$e_i^{'}$ within range of 25% of e_i?
Project 1			
19	25 ± 6.25	25	√
20	27 ± 6.75	26.23	√
21	30 ± 7.5	27.53	√
22	33 ± 8.25	28.83	√
23	34 ± 8.5	30.10	√
24	35 ± 8.75	31.28	√
25	36 ± 9	32.38	√
26	40 ± 10	33.44	√
27	40 ± 10	34.51	√
28	41 ± 10.25	35.58	√
Project 2			
23	69 ± 17.25	75.82	√
24	70 ± 17.5	77.30	√
25	74 ± 18.5	74.69	√
26	78 ± 19.5	**76.40**	√
27	79 ± 19.75	84.14	√
28	83 ± 20.75	88.64	√
29	85 ± 21.25	94.28	√
30	93 ± 23.25	96.48	√
31	102 ± 25.5	**93.36**	√
32	109 ± 27.25	**102.56**	√
33	110 ± 27.5	**102.91**	√
Project 3			
21	153 ± 38.25	**148.54**	√
22	162 ± 40.5	**159.07**	√
23	173 ± 43.25	**167.06**	√
24	180 ± 45	**174.67**	√
25	184 ± 46	**181.04**	√
26	190 ± 47.5	**189.07**	√
27	196 ± 49	196.18	√
28	204 ± 51	**203.80**	√
29	208 ± 52	**207.65**	√
30	210 ± 52.5	216.32	√

In all the data sets, the measurement $pred(0.25) \geq 0.75$ holds true. The bold values in Table 3.4 illustrate the cases when the model underestimates the actual fault count data.

We also calculated the MMRE for each of the data sets. The MMRE values for the three data sets were 0.0992, 0.06558 and 0.0166, respectively. Each of these values satisfy the criterion of MMRE ≤ 0.25, therefore we have confidence that we have a good set of predictions. For evaluating the prediction stability, we calculated whether the prediction in week i is within 10% of the prediction in week $i-1$. The results (Table 3.5) indicate that the predictions are indeed stable.

Table 3.5: Testing for prediction stability.

Week i	Prediction in week i	10% of the prediction in week $i-1$	Prediction stability
Project 1			
19	25	–	–
20	26.23	25 ± 2.5	√
21	27.53	26.23 ± 2.62	√
22	28.83	27.53 ± 2.75	√
23	30.10	28.83 ± 2.88	√
24	31.28	30.10 ± 3.01	√
25	32.37	31.28 ± 3.12	√
26	33.44	32.37 ± 3.23	√
27	34.50	33.44 ± 3.34	√
28	35.57	34.50 ± 3.45	√
Project 2			
23	75.81	–	–
24	77.30	75.81 ± 7.58	√
25	74.69	77.30 ± 7.73	√
26	76.39	74.69 ± 7.46	√
27	84.14	76.39 ± 7.63	√
28	88.64	84.14 ± 8.41	√
29	94.28	88.64 ± 8.86	√
30	96.48	94.28 ± 9.42	√
31	93.35	96.48 ± 9.64	√
32	102.56	93.35 ± 9.33	√
33	102.91	102.56 ± 10.25	√
Project 3			
21	148.53	–	–
22	159.06	148.53 ± 14.85	√
23	167.06	159.06 ± 15.90	√
24	174.66	167.06 ± 16.70	√
25	181.04	174.66 ± 17.46	√
26	189.07	181.04 ± 18.10	√
27	196.18	189.07 ± 18.90	√
28	203.80	196.18 ± 19.61	√
29	207.65	203.80 ± 20.38	√
30	216.32	207.65 ± 20.76	√

The results of applying $pred(l)$, MMRE and the measure of prediction stability show that the GP model is able to produce significantly accurate predictions. We can, thus reject the null hypothesis, $H_{0-\text{acc}}$ in favor of the alternative, $H_{1-\text{acc}}$.

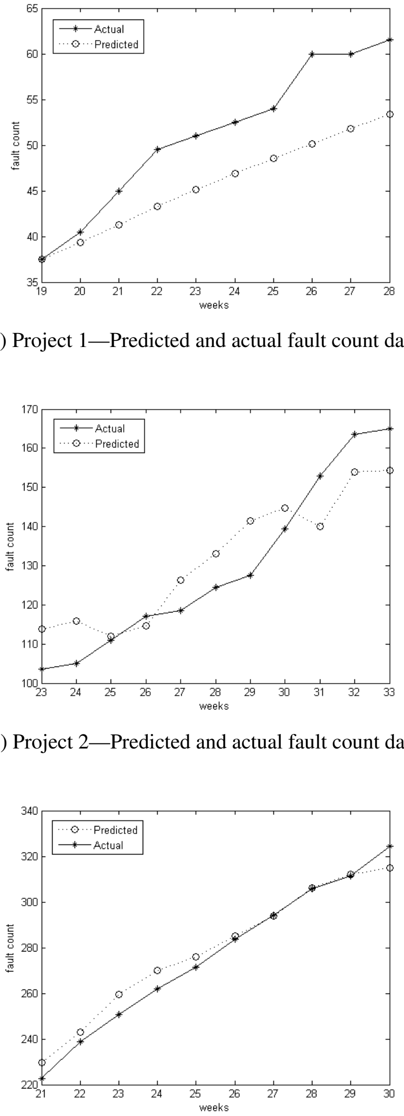
Figure 3.6 shows the comparison of actual and predicted fault count data for the three projects. The actual and predicted fault count data is multiplied by a constant factor due to proprietary concerns. The difference between the actual and predicted fault count is the least for data from Project 3, which also has the best MMRE value of 0.0166. These charts show that the GP evolved curve is able to learn the pattern in failure count data and adapts reasonably well.

3.5.4 Summary of results

The hypothesis tested if GP could be a suitable approach for evolving a SRGM based on fault count data. The results of applying the evaluation criteria, as described in Section 3.5.1, confirmed that GP represents a suitable approach for modeling software reliability growth based on fault count data, both in terms of goodness of fit and predictive accuracy. In terms of goodness of fit, the K-S test statistic for all three data sets showed that at significance level of 0.05, the GP model fits well to the set of observations. We also calculated the Spearman's rank correlation coefficient to determine the strength of the relationship between actual values and and estimated model values. The results showed that at significance level of 0.05, there exists a strong relationship between the two distributions. The results obtained are also promising in terms of predictive accuracy. The custom measures of MMRE ≤ 0.25 and $pred(0.25) \geq 0.75$, as indicative of a good prediction system, holds true in all the three data sets. However, we noted a considerable variation in MMRE values for the three validation data sets. This indicates the sensitivity of GP to changes in the training set and is indicative of the adaptive nature of the GP algorithm to deal with heterogeneous data. To have a degree of confidence about the accuracy of future estimates, we resorted to a good rule of thumb for evaluating predictive stability (see Section 3.5.1), which also gave results in support of GP.

3.6 Study stage 3: Comparative evaluation with traditional SRGMs

In this stage, we present the results of comparison between models evolved using GP and three other traditional SRGMs based on the same data as in Stage 2. We discuss the



(a) Project 1—Predicted and actual fault count data.

(b) Project 2—Predicted and actual fault count data.

(c) Project 3—Predicted and actual fault count data.

Figure 3.6: Actual and predicted fault count data for three projects.

selection of traditional SRGMs, hypotheses, the evaluation measures and the results. We do not discuss the experimental setup as it is the same as for Stage 2 (Section 3.5.2).

3.6.1 Selection of traditional SRGMs

Since we are interested in comparing predictions of weekly fault count data, we select three traditional SRGMs that represent the fault count family of models [108]. These three models are Goel-Okumoto non-homogeneous Poisson process model (GO-NHPP) [109], Brooks and Motley's Poisson model (BM) [52] and Yamada's *S*-shaped growth model (YAM) [347]. We selected them because these models present a fair representation of the fault count family of models and, in addition, represent different forms of growth curves. In particular, GO-NHPP and BM are concave (or exponential) while YAM is *S*-shaped. Also we had limitations in terms of information requirements of certain models, so they were not selected for comparison, like Shooman's exponential model and its hazard function requiring knowing the parameters of the total number of instructions in the program and debugging time since the start of system integration [108].

3.6.2 Hypothesis

In order to formalize the purpose of this experiment, we define the following hypotheses:

$H_{0-\text{val}}$: The predictions of the GP evolved model are not significantly more valid as compared with traditional models.

$H_{1-\text{val}}$: The predictions of the GP evolved model are significantly more valid as compared with traditional models.

$H_{0-\text{gof}}$: The GP evolved model does not give significantly higher goodness of fit as compared with traditional models.

$H_{1-\text{gof}}$: The GP evolved model gives significantly higher goodness of fit as compared with traditional models.

$H_{0-\text{res}}$: There is no significant difference between the residuals of the GP evolved model as compared with traditional models.

$H_{1-\text{res}}$: There is a significant difference between the residuals of the GP evolved model as compared with traditional models.

In order then to test the above hypotheses, we use different evaluation measures as detailed in the next section.

3.6.3 Evaluation measures

It is usually recommended to use more than one measure to determine model applicability (see e.g., [257]) because reliance on a single measure can lead to making incorrect choices. We used measures of model validity, goodness of fit and distribution of residuals to compare the GP evolved model with traditional reliability growth models.

Model validity is measured in terms of prequential likelihood ratio (PLR), the Braun statistic and the adjusted mean square error (AMSE). The PLR of two prediction systems, A and B, is the running product of ratio of their successive on-step ahead predictions $\hat{f}_j^A(t_j)$ and $\hat{f}_j^B(t_j)$ respectively [51]:

$$PLR_i^{AB} = \prod_{j=s}^{j=i} \frac{\hat{f}_j^A(t_j)}{\hat{f}_j^B(t_j)}$$

In our case, we select the actual time distribution of weekly fault count data as a reference and conduct pair-wise comparisons of all other models' predictions against it. Then the model with the relatively smallest prequential likelihood ratio can be expected to provide the most trustworthiest predictions. (For further details on PLR please see [1, 51].) We complement the measure of prequential likelihood ratio with two measures of variability, namely the Braun statistic and AMSE. The Braun statistic can be used to measure the accuracy of fault count predictions and is given by the following formula [51]:

$$\text{Braun statistic}\{\hat{E}[N_k]; k = s,\ldots,r\} = \frac{\sum\limits_{k=s}^{r}(n_k - \hat{E}[N_k])^2 x_k}{\sum\limits_{k=s}^{r}(n_k - \bar{n})^2 x_k}$$

Where n_k is the actual fault count within successive time intervals, $x_k, k = s,\ldots,r$. $\hat{E}[N_k]$ represents the predicted fault count data and $\bar{n}$ represents the mean of the actual fault count data. AMSE is a simple measure based on the mean square error which takes into account the mean of the data sets and is given by the following formula [56]:

$$AMSE = \sum_{i=1}^{i=n} \frac{(E_i - \hat{E}_i)^2}{(\bar{E}_i) * \bar{\hat{E}}_i)^2}$$

where E_i is the actual fault count data and $\hat{E}_i$ is the predicted fault count data.

To measure a particular model's bias, we examine the *distribution of residuals* to compare models as suggested in [193, 270]. The model's *goodness of fit* in our case was measured using the Kolmogorov-Smirnov (K-S) test [137]. For the K-S test, we use $\alpha = 0.05$ and if the K-S statistic J is greater or equal than the critical value J_α, we infer that the two samples did not have the same probability distribution and hence do not represent significant goodness of fit, i.e., the null hypothesis of two samples having the same probability distribution, was rejected in favor of the alternate hypothesis.

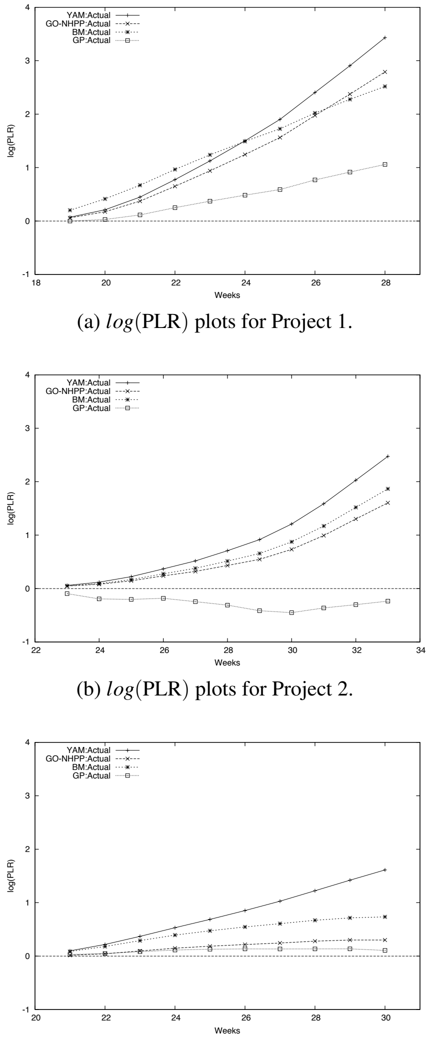
3.6.4 Results

Figure 3.7 shows the PLR analysis for the three data sets. The $log(\text{PLR})$ of actual time distribution of weekly fault count data is chosen as the the reference; and it is indicated as a straight line in the plots of Figure 3.7. It can be seen that the curve for the PLR of the GP model with the actual fault count data (GP:Actual) is closer to the straight line as compared with the same curves for the traditional models; confirming that GP predictions are better in modeling reality as compared with traditional reliability growth models.

The variability measures of Braun statistic and AMSE obtained for each data set of all models were compared using matched paired two-sided t-test at significance level, $\alpha = 0.1$. We compared the variability measures of the GP model with each of the traditional models. The null hypothesis was formulated as that there was no difference between the variability statistics of GP and that of the particular traditional model under comparison. The alternate hypothesis to test was then that there existed such a difference. Using normal quartile plot of the samples' variability differences to assess any radical departures from the normal distribution showed that they had approximately normal distribution. The results of applying the matched paired two sided t-test are shown in Table 3.6.

Table 3.6: Statistical results for Braun statistic and AMSE.

Comparative models	t-statistic
Braun statistic, t_α=±2.42	
GP:BM	−3.97
GP:YAM	−4.80
GP:GO-NHPP	−1.64
AMSE statistic, t_α=±2.42	
GP:BM	−1.23
GP:YAM	−1.39
GP:GO-NHPP	−1.03



(a) $log(\mathrm{PLR})$ plots for Project 1.

(b) $log(\mathrm{PLR})$ plots for Project 2.

(c) $log(\mathrm{PLR})$ plots for Project 3.

Figure 3.7: $log(\mathrm{PLR})$ plots for three projects.

The critical values of t for α=0.1 and degrees of freedom $n-1$ is $t_\alpha = \pm 2.92$. If the calculated t-statistic lied in the critical region, we were able to reject the null hypothesis of no difference between the samples.

We can observe from Table 3.6 that there is a statistical difference between GP and two of the traditional models (BM and YAM) for the Braun statistic. However, for the AMSE statistic, there is no statistical difference between GP and traditional models. This shows that the GP model, while optimizing the Braun statistic, degrades AMSE. This result strengthens the viewpoint of Mair et al. [227] that using a fitness function for GP that is not specifically tied to a single measure, but takes into account multiple objectives, may give overall better results for the GP model. Based on the results of applying PLR, Braun statistic and AMSE, we are not able to reject the null hypothesis, $\text{H}_{0-\text{val}}$ in support of the alternative hypothesis, $\text{H}_{1-\text{val}}$.

Table 3.7 shows the statistic J for the two sample K-S test performed on the validation fault count data and the predictions by the GP and traditional SRGMs.

Table 3.7: Results of applying K-S test.

	J_{GP}	J_{BM}	J_{YAM}	$J_{GO-NHPP}$
Proj. 1, J_α=0.70	0.40	0.70	1.00	0.8
Proj. 2, J_α=0.64	0.27	0.73	0.82	0.54
Proj. 3, J_α=0.70	0.10	0.30	0.70	0.20

For Project 1, we see that $J_{GP} < J_\alpha$, suggesting that the predicted fault count data, as provided by the GP model, fits quite well to the set of observations. On the other hand, the J statistic for all other traditional models are either equal to or greater than J_α. For Project 2, the GP model along with the GO-NHPP model have K-S statistic J less than J_α and, finally, for Project 3, the GP model along with BM and GO-NHPP provide K-S statistic J less than J_α.

While we see the traditional models giving statistically significant goodness of fit for Project 2 and 3 on three occasions, neither of them gave statistics that were lower than the corresponding K-S statistic for the GP model. This is, however, not enough to reject the null hypothesis $\text{H}_{0-\text{gof}}$, i.e., we are inconclusive regarding the significance of the goodness of fit of competing models.

Figure 3.8 shows the box plots of the residuals for all the models for the three projects. For Project 1 (Figure 3.8a), all the box plots show the tendency of underestimating; with the length of the box and tails of the GP model and BM model being smaller, indicating that the prediction bias is not severe. The tendency of the GP model in case of Project 2 (Figure 3.8b) is to overestimate but the bias is smaller as compared to other models. In case of Project 3 (Figure 3.8c) all box plots represent a tendency to underestimate while the GP model presents relatively less bias with residuals both above and below 0.

Since the box plots in Figure 3.8 are not significantly skewed, we applied matched paired t-tests of the residuals for each data set to compare the GP model with each of the traditional models. The results are presented in Table 3.8 and show that the residuals from the GP model are significantly different and less variable from the residuals for traditional models for each data set at α=0.05. Therefore, we are able to reject the null hypothesis, $\mathrm{H}_{0-\mathrm{res}}$ in support of the alternative hypothesis, $\mathrm{H}_{1-\mathrm{res}}$.

Table 3.8: **t**-test results for residuals.

	$t_{GP:BM}$	$t_{GP:YAM}$	$t_{GP:GO-NHPP}$
Proj. 1, t_α=±2.42	−32.18	−6.42	−6.59
Proj. 2, t_α=±2.23	−7.76	−7.11	−7.53
Proj. 3, t_α=±2.26	−23.43	−7.92	−4.56

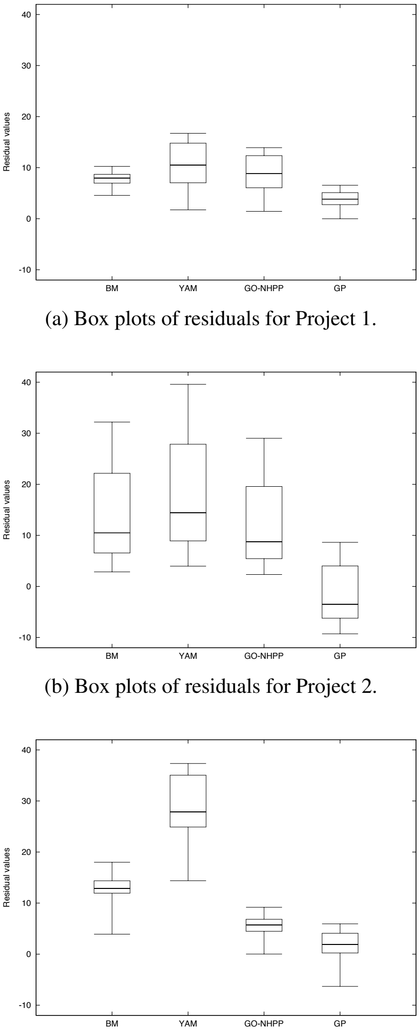
3.6.5 Summary of results

Stage 3 of the study presented the results of a comparative evaluation of fault count data predictions from models evolved by genetic programming and traditional reliability growth models. The results have been evaluated in terms of model validity, goodness of fit and distribution of residuals. For evaluating model validity, the results of using prequential likelihood ratio show favorability concerning the GP model. However, the results of AMSE and Braun statistic did not show a statistically significant difference between the GP model and traditional software reliability growth models for all the projects.

The goodness of fit of GP models was not found to be significantly higher than all the models for the three data sets; so we remain inconclusive regarding the significance of goodness of fit. The visual inspection of the box plots of residuals and matched paired t-tests showed the GP model predictions to be less biased than traditional models. The evaluation results show that prediction of fault count data using genetic programming is a promising approach.

3.7 Empirical validity evaluation

There can be different threats to the validity of experimental results in Stage 2 and 3 of this study. *Conclusion validity* refers to the statistically significant relationship be-



(a) Box plots of residuals for Project 1.

(b) Box plots of residuals for Project 2.

(c) Box plots of residuals for Project 3.

Figure 3.8: Charts showing box plots of residuals for three projects.

tween the treatment and the outcome [342]. One of the threats to conclusion validity is the use of MMRE in Stage 2 of the study which has been criticized in [105] for being unreliable. We, however, used an additional measure (Spearman's rank correlation coefficient) for measuring the strength of relationship to minimize this threat. A similar threat is that we might have missed applying a more suitable evaluation measure. However, to our knowledge, the evaluation measures used in the study reflect the ones commonly used for evaluating prediction models. *Internal validity* refers to a causal relationship between the treatment (independent variable) and outcome (dependent variable) [342]. Threats to internal validity are reduced in several ways. First, the splitting of data sets into training and testing sets were always done using the rule that first 2/3 of the data set is used for training, while the rest 1/3 of the data set is used for testing purposes. There were two reasons for persisting with this choice. First of all, this choice of splitting is commonly used in many machine-learning studies [341]. Secondly, since the fault count histories are time-series data, it is logical to choose a split that preserves the chronological time series occurrences of faults. Another possible threat to internal validity was minimized by not pre-processing the data before applying any technique, except that the data were aggregated on weekly/monthly basis due to the availability of data sets in this format. *Construct validity* is concerned with the relationship between theory and observation [342]. The different evaluation measures used in Stage 2 and Stage 3 of this study reflect the construct under study, e.g., Kolmogorov-Smirnov test is used for measurement of goodness of fit, which is a commonly used test for this measure. Other measures used in this study also relate to the measurement of a specific property. *External validity* is concerned with generalization of results outside the scope of the study. The experiments in Stage 2 and Stage 3 of this study are conducted on three different data sets taken from an industrial setting. However, these projects are carried out by one organization following similar development methods. The generalizability of the research can be improved by experimenting with data sets taken from diverse projects employing different development methodologies.

3.8 Discussion

This chapter presented a multi-stage exploration of using GP for the purpose of software fault prediction. Stage 1 discussed the mechanisms enabling GP to potentially be an effective modeling technique. Stage 2 presented an experiment where we applied GP to evolve models based on weekly fault count data. Stage 3 presented the results of comparing models evolved using GP with three other traditional SRGMs based on the same data sets as in Stage 2.

In our case, we had one independent and one dependent variable. Hence, the GP algorithm generated good models efficiently within the termination criterion of reasonable number of generations. However, it is common that efficiency and effectiveness of GP drops if the data tables contain hundreds of variables as the GP algorithm then can take a considerable amount of time in isolating the key features [272].

While measures of goodness of fit and predictive accuracy are important, we agree with Mair et al. [227] that these measures are not enough to constitute a useful prediction system. Therefore, the explanatory value (transparency of solution) and ease of configuration are also important aspects that require discussion. Since the output of a GP system is an algebraic expression, it has the potential of generating transparent solutions; however, the solutions can become complex as the number of nodes in the solution increases. There is a trade-off in having more accurate predictions and less simplicity of the algebraic expressions but we believe that this trade-off is manageable as achieving accurate models within acceptable thresholds is possible. In terms of ease of configuration, we found that configuring GP control parameters requires considerable effort. Different facets need to be determined, e.g., evaluation function, genetic operators and probabilities, population size and termination criterion to name a few. The parameter tuning problem is time consuming because the control parameters are not independent but interact in complex ways and trying all possible combinations of all parameters is practically infeasible [272].

3.9 Summary of the chapter

The overall contribution of this chapter is exploring the GP mechanism that might be suitable for modeling, empirically investigating the use of GP as a potential prediction tool in software V&V while, at the same time, performing a comparative evaluation of GP with traditional software reliability growth models. Stage two of this study evaluated the GP-evolved models in terms of goodness of fit and predictive accuracy. For evaluating goodness of fit, the K-S statistic and Spearman's rank correlation coefficient gives statistically significant results in favor of adaptability of GP evolved model. The resulting statistics for evaluating predictive accuracy are also encouraging with $pred(0.25)$, MMRE and measure of prediction stability offering results in favor of statistically significant prediction accuracy. In Stage 3 of the study, the results have been evaluated in terms of model validity, goodness of fit and distribution of residuals. For evaluating model validity, although the results of using prequential likelihood ratio show favorability to the GP model, the same is not the case with the Braun statistic and AMSE. The GP model was also found to have either an equivalent or better goodness of fit as compared to traditional models, but not statistically significant in every case. The visual inspection of the box plots of residuals and matched paired t-tests showed the GP model predictions to be less biased than traditional models.

These early results of using a search-based technique for software fault prediction are carried forward in the next chapter, Chapter 4, which investigates cross-release prediction of fault-count data from large and complex industrial and open source software projects.

Chapter 4

Empirical evaluation of cross-release fault count predictions in large and complex software projects

Originally published as a book chapter with the title — *Genetic programming for cross-release fault count predictions in large and complex software projects*, in the book — *Evolutionary computation and optimization algorithms in software engineering* — published by IGI Global.

W. Afzal, R. Torkar, R. Feldt and T. Gorschek

4.1 Introduction

One influential factor in software quality are the number of faults incurred during the development life cycle which can have direct impact on costs. Software verification and validation activities constitutes a fair percentage of the total software life cycle cost; some say around 40% [36] and, hence, efficient resource allocation for quality assurance activities is required. Thus, fault prediction models have attracted considerable interest (as will be shown in Section 4.2), both in research and in practice. From a research point of view, new methods of fault prediction are regularly being proposed, and their predictability assessed, at varying levels of detail. The practical aspect of such models has strong implications on the quality of the software a project develops since the information gained from such models can e.g., be an important decision-making tool for project managers.

The number of faults in a software component, or in a particular release of software, represents indirect quantitative measures of software quality. A fault prediction model uses historic software quality data in the form of metrics (including software fault data) to predict the number of software faults in a component or a release [178]. Fault predictions for a software release are fundamental to the efforts of quantifying software quality. A fault prediction model helps a software development team in prioritizing the effort spent on a software project. If the predictions forecasts a high number of faults in the coming release of a project, management has the option of investing required levels of effort to circumvent possible project failures.

This chapter presents both quantitative and qualitative evaluations for cross-release predictions of fault count data gathered from both open source and industrial software projects. Fault counts denotes the cumulative faults aggregated on a weekly or monthly basis. We quantitatively compare the results from traditional and machine learning approaches to fault count predictions and also assess various qualitative criteria for better trade-off analysis. The main purpose is to increase empirical knowledge concerning innovative ways of predicting fault count data and to apply the resulting models in a manner, which is suited to multi-release software development projects.

Linear regression is a typical method used for software fault predictions; however this may not be the best approach. This argument is supported by the fact that software engineering data come with certain characteristics that creates difficulties in making accurate software prediction models. These characteristics include missing data, large number of variables, strong co-linearity between the variables, heteroscedasticity[1], complex non-linear relationships, outliers and small sample size [113]. Therefore, it is not surprising that we possess an incomplete understanding of the phenomenon

[1] A sequence of random variables with different variances.

under study since *it is very difficult to make valid assumptions about the form of the functional relationship between the variables* [46]. This argument strengthens earlier established results that show program metrics begin insufficient for accurate prediction of faults. Moreover, the acceptability of models has seen little success due to lack of meaningful explanation of the relationship among different variables and the lack of generalizability of model results [113]. Applications of computational and artificial intelligence have attempted to deal with some of these challenges, see e.g., [354], mainly because of their inherent intelligent modeling mechanisms to deal with data. There are several reasons for using these techniques for fault prediction modeling:

1. They do not depend on assumptions about data distribution and relationship between independent and dependent variables.

2. They are independent of any assumptions about the stochastic behavior of software failure process and the nature of software faults [306].

3. They do not conceive a particular structure for the resulting model.

4. The model and the associated coefficients can be evolved based on the fault data collected during the initial test phase.

While the use of artificial intelligence and machine learning is applied with some success in software reliability growth modeling and software fault predictions, only a small number of these studies make use of data from large industrial software projects, see e.g., [313]. Performing large empirical studies is hard due to difficulties in getting necessary data from large software projects, but if we want to generalize the use of some technique or method, larger type software need to be investigated to gain better understanding. Moreover, due to the novelty of applying artificial intelligence and machine learning approaches, researchers many times focus more on introducing new approaches, validated on a smaller scale, than validating existing approaches on a larger scale. In this chapter we try to focus on the latter.

Another dimension that lacks researchers' attention is cross-release prediction of faults. With the growing adoption of agile software development methodologies, prediction of faults in subsequent releases of software will be an important decision tool. With short-timed releases, the software development team might not be inclined towards gathering many different program metrics in a current release of a project. Therefore, machine learning techniques can make use of less and commonly used historical data to become a useful alternative in predicting the number of faults across different releases of a software project.

The goals of this study differ in some important ways from related prior studies (as will be covered in detail in Section 4.2). Our main focus is on evaluating a variety of techniques for cross-release prediction of fault counts, on data sets from large projects; to our knowledge this is novel. We evaluate the created models on fault data from several large software projects, some from open-source and some from industry (see Section 4.3).

Our study is also unique in comparing multiple different fault count modeling techniques, both traditional and several machine learning approaches. The traditional approaches we have selected are three software reliability growth models (SRGMs) that represent the fault count family of models [108]. These three models are Goel-Okumoto non-homogeneous Poisson process model (GO) [109], Brooks and Motley's Poisson model (BMP) [52] and Yamada's *S*-shaped growth model (YAM) [347]. We selected them because these models provide a fair representation of the fault count family of models (representing different forms of growth curves). In particular, GO and BMP are concave (or exponential) while YAM is *S*-shaped. We also include a simple and standard least-squares linear regression as a baseline.

The machine learning approaches we compare with are genetic programming (GP), artificial neural networks (ANN) and support vector machine regression (SVM). We selected these because they are very different/disparate and have seen much interest in the machine learning (ML) communities of late, see e.g., [172, 184, 316] for some examples.

Our main goal is to answer the question:

> Is GP a better approach for cross-release prediction of fault counts on fault data from large software projects in comparison with traditional and machine learning approaches?

To answer it we have identified a number of more detailed research questions listed in Section 4.4. By applying the model creation approaches described above and by answering the research questions this chapter presents the following results:

1. Quantitative and qualitative assessment of the generalizability and applicability of different modeling techniques by the use of extensive data sets covering both open source and industrial software projects.

2. Comparative evaluations with both traditional and machine learning models for *cross-release* prediction of fault count data.

The remainder of this chapter is organized as follows. In Section 4.2, we present the background for this study. Section 4.3 elaborates on the data collection procedure. Section 4.4 describes the research questions, while Section 4.6 provides a brief introduction to the techniques used in the study. Section 4.5 describes the different evaluation measures used in the study while Section 4.7 covers the application of different techniques and the corresponding evaluation. The validity evaluation is presented in Section 4.8, while discussion and conclusions are presented in Section 4.9.

4.2 Related work

The research into software quality modeling based on software metrics is *used to predict the response variable which can either be the class of a module (e.g., fault-prone and not fault-prone) or a quality factor (e.g., number of faults) for a module* [179]. There have been a number of software fault prediction and reliability growth modeling techniques proposed in software engineering literature. The applicable methods include statistical methods (random-time approach, stochastic approach), machine learning methods and mixed algorithms [67]. Despite the presence of large number of models, there is no agreement within the research community about the best model. One of the reasons for this situation is that models exhibit different predictive accuracies across different data sets. Therefore, the quest for a consistently accurate predictor model is continuing and the current result is that the prediction problem is seen as being largely unsolvable and NP-hard [67, 290]. Due to a large number of studies covering software quality modeling (for both classifying fault-proneness and predicting software faults), the below references should be seen more as representative than exhaustive.

Gao and Khoshgoftaar [107] empirically evaluated eight statistical count models for software quality prediction. They showed that with a very large number of zero response variables, the zero inflated and hurdle-count models are more appropriate. The study by Yu et al. [352] used number of faults detected in earlier phases of the development process to predict the number of faults later in the process. They compared linear regression with a revised form of, an earlier proposed, Remus-Zilles model. They found a strong relationship between the number of faults during earlier phases of development and those found later, especially with their revised model. Khoshgoftaar et al. [176] showed that the typically used least squares linear regression and least absolute value linear regression do not predict software quality well when the data does not satisfy the normality assumption and thus two alternative parameter estimation procedures (relative least square and minimum relative error) were found more suitable in this case. In [245], the discriminant analysis technique is used to classify the programs into either fault-prone and not fault-prone based upon the uncorrelated measures of program complexity. Their technique was able to yield less Type II errors (mistakenly classifying a fault-prone component as fault-prone) on data sets from two commercial systems.

In [45], optimized set reduction classifications (that generates logical expressions representing patterns in the data) were found to be more accurate than multivariate logistic regression and classification trees in modeling high-risk software components. The less optimistic results of using logistic regression are not in agreement with Khoshgoftaar's study [168], which supports using logistic regression for software quality classification. Also the study by Denaro et al. [84] used logistic regression to successfully classify faults across homogeneous applications. Basili et al. [31] verified that most of Chidamber and Kemerer's object-oriented metrics are useful quality indicators for fault-prone classes. Ohlsson et al. [260] investigated the use of metrics for release n to identify the most fault-prone components in release $n+1$. Later, in [261], principal component analysis and discriminant analysis was used to rank the software components in several groups according to fault-proneness.

Using the classification and regression trees (CART) algorithm, and by balancing the cost of misclassification, Khoshgoftaar et al. [170] showed that the classification-tree models based on several product, process and execution measurements were useful in quality classification for successive software releases. Briand et al. [49] proposed multivariate adaptive regression splines (MARS) to classify object-oriented classes as either fault-prone or not fault-prone. MARS outclassed logistic regression with an added advantage that the functional form of MARS is not known *a priori*. In [235], the authors show that static code attributes like McCabe's and Halstead's are valid attributes for fault prediction. It was further shown that naïve Bayes outperformed the decision-tree learning methods.

As discussed briefly in Section 4.1, the use of regression analysis might not be the best approach for software fault prediction. And, hence, we find numerous studies making use of machine intelligence techniques for software fault prediction. Applications of artificial neural networks to fault predictions and reliability growth modeling mark the beginning of several studies using machine learning for approximations and predictions. *Neural networks have been found to be a powerful alternative when noise in the input-generating process complicates the analysis, a large number of attributes describe the inputs, conditions in the input-generating process change, available models account for some but not all of the data, the input-generating distribution is unknown and probably non-Gaussian, it is expensive to estimate statistical parameters, and nonlinear relationship are suspected* [58]. These characteristics are also common to data collected from a typical software development process. Karunanithi et al. published several studies [163, 164, 165, 166, 167] using neural network architectures for software reliability growth modeling. Other examples of studies reporting encouraging results include [3, 19, 86, 117, 118, 135, 169, 177, 182, 184, 293, 311, 312, 313, 314]. While, finally, Cai et al. [59] observed that the prediction results of ANNs show a positive overall pattern in terms of probability distribution but were found to be poor at quantitatively estimating the number of software faults.

A study by Gray et al. [113] showed that neural network models show more predictive accuracy as compared with regression based methods. The study also used a criteria-based evaluation on conceptual requirements and concluded that not all modeling techniques suit all types of problems. CART-LAD (least absolute deviation) performed the best in a study by Khoshgoftaar et al. [179] for fault prediction in a large telecommunications system in comparison with CART-LS (least squares), *S*-plus, regression tree algorithm, multiple linear regression, artificial neural networks and case-based reasoning.

Gyimóthy et al. [121] used OO metrics for predicting the number of faults in classes using logical and linear regression, decision-tree and neural network methods. They found that the results from these methods were nearly similar. A recent study by Lessman et al. [209] also concluded that, with respect to classification, there were no significant differences among the top-17 of the classifiers used for comparison in the study.

Apart from artificial neural networks, some authors have proposed using fuzzy models, as in [60, 61, 297, 323], and support vector machines, as in [316], to characterize software reliability.

In the later years, interest has shifted to evolutionary computation approaches for software reliability growth modeling. Genetic programming has been used for software reliability growth modeling in several studies [6, 8, 9, 73, 75, 262, 355]. The comparisons with traditional software reliability growth models indicate that genetic programming may have an edge with respect to predictive accuracy and also does not need assumptions that are common in the traditional models. There are also several studies where genetic programming has been successfully used for software quality classification [172, 173].

There are also studies that use a combination of techniques, e.g., [316], where genetic algorithms are used to determine an optimal neural network architecture and in [255], where principal component analysis is used to enhance the performance of neural networks.

As mentioned in Section 4.1, very few studies have looked at cross-release predictions of fault data on a large scale. Ostrand and Weyuker [264] presented a case study using 13 releases of a large industrial inventory tracking system. Among several goals of that study, one was to investigate the fault persistence in the files between releases. The study concluded with moderate evidence supporting that files containing high number of faults in one release remain ‘high fault files’ in later releases. The authors later extend their study in [265] by including four further releases. They investigated which files in the next release of the system were most likely to contain the largest number of faults. A negative binomial regression model was used to make accurate predictions about expected number of faults in each file of the next release of a system.

4.3 Selection of fault count data sets

We use fault count data from two different types of software projects: open source software and industrial software. For all of these projects we have data for multiple releases of the same software system. Between releases there can be both changes and improvements to existing functionality as well as additions of new features. The software projects together represent many man years of development and span a multitude of different software applications targeting e.g., home users, small-business users and industrial, embedded systems.

The included open source systems are: Apache Tomcat[2], OpenBSD[3] and Mozilla Firefox[4]. Apache Tomcat is a servlet container implementing the Java servlet and the JavaServer Pages. Members of the Apache Software Foundation (ASF), and others, contribute in developing Apache Tomcat. OpenBSD is a UNIX-like operating system developed at the University of California, Berkley. OpenBSD supports a variety of hardware platforms and includes several extra security options like built-in cryptography. Mozilla Firefox is an open-source web-browser from the Mozilla Corporation, supporting a variety of operating systems.

In the following, the fault count data from these open source software projects are referred to as OSStom, OSSbsd and OSSmoz, respectively.

The industrial fault count data sets come from three large companies specializing in different domains. The first industrial data set (IND01) is from a European company in the space industry. The multi-release software is for an on-board computer used in a satellite system. It consists of about $70,000$ lines of manually written C code for drivers and other low-level functions and about $230,000$ lines of C code generated automatically from Simulink models. The total number of person hours used to develop the software is on the order of $30,000$. About 20% of this was spent in system testing and 40% in unit testing. The faults in the data set is only from system testing, the unit testing faults are not logged but are instead corrected before the final builds.

The second and third fault count data sets (IND02 and IND03) are taken from a power and automation company specializing in power products, power systems, automation products, process automation and robotics. IND02 comes from one of their robotic controller software that makes use of advanced motion technology to program robot systems. This software makes use of a state-of-the-art self-optimizing motion technology, security and error handling mechanism and advanced user-authorization system. IND03 consists of fault count data from robotic packaging software. This software comes with an advanced vision technique and integrated conveyor tracking

[2] http://tomcat.apache.org

[3] http://www.openbsd.org

[4] http://www.mozilla.com

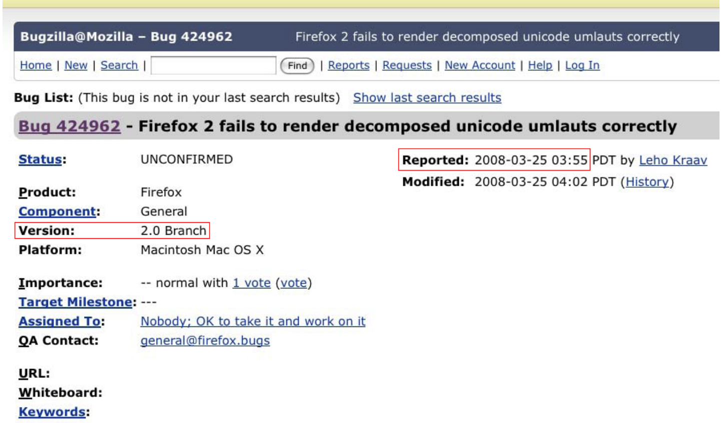
capability; while being open to communicate with any external sensor. The total number of person hours used to develop the two projects is on the order of $2,000$.

The last data set, IND04, comes from a large mobile hydraulics company specializing in engineered hydraulic, electric and electronic systems. The fault count data set comes from one of their products, a graphical user interface integrated development environment, which is a part of a family of products providing complete vehicle control solutions. The software allows graphical development of machine management applications and user-specific service and diagnostic tools. The software consists of about $350,000$ lines of hand written Delphi/Pascal code (90%) and C code (10%). Total development time is about $96,000$ person hours, 30% of this has been on system tests.

4.3.1 Data collection process

The fault count data from the three open source projects: Apache Tomcat (OSStom), OpenBSD (OSSbsd) and Mozilla Firefox (OSSmoz), come from web-based bug reporting systems.

As an example, Figure 4.1 shows a bug report for Mozilla Firefox. For OSStom and OSSmoz, we recorded the data from the 'Reported' and 'Version' fields as shown in the Figure 4.1. For OSSbsd, the data was recorded from the 'Environment' and 'Arrival-Date' fields of the bug reports. We include all user submitted bug reports in


Home | New | Search | Find | Reports | Requests | New Account | Help | Log In

Bugzilla@Mozilla – Bug 424962 Firefox 2 fails to render decomposed unicode umlauts correctly

Bug List: (This bug is not in your last search results) Show last search results

Bug 424962 - Firefox 2 fails to render decomposed unicode umlauts correctly

Status: UNCONFIRMED

Reported: 2008-03-25 03:55 PDT by Leho Kraav

Modified: 2008-03-25 04:02 PDT (History)

Product: Firefox

Component: General

Version: 2.0 Branch

Platform: Macintosh Mac OS X

Importance: -- normal with 1 vote (vote)

Target Milestone: ---

Assigned To: Nobody; OK to take it and work on it

QA Contact: general@firefox.bugs

URL:

Whiteboard:

Keywords:

Figure 4.1: A sample bug report.

our data collection because the core development team examines each bug report and decides upon a course to follow [210]. The severity of the user submitted faults was not considered as all submitted bug reports were treated equally. A reason for treating all user submitted bug reports as equal was to eliminate inaccuracy and subjective bias in assigning severity ratings.

Concerning the industrial software, we were assisted by our industrial partners in provision of the fault count data sets IND01–IND04. Table 4.1 show more details regarding the data collected from the open source and industry software projects. The data sets were impartially split into training and test sets. In line with the goals of the study (i.e., cross-release prediction), we used a finite number of fault count data from multiple releases as a training set. The resulting models were evaluated on a test set, comprising of fault count data from subsequent releases of respective software projects. The length of the test sets also determined the prediction strength, x time units into future, where x equals the length of the test set and is different for different data sets. We used the cumulative weekly count of faults for all the data sets, except for IND02 and IND03 for which the monthly cumulative counts were used due to the availability of the fault data in monthly format.

Table 4.1: Data collection from open source and industrial software projects, time span mentioned in () in the second column is same for the releases preceding.

Software	Data collected from releases and time span	Training and test sets	Length of training set	Lenght of testing set
OSStom	6.0.10, 6.0.11, 6.0.13 (Mar.–Aug. 2007), 6.0.14 (Aug.–Dec. 2007)	Train on 6.0.10, 6.0.11, 6.0.13 Test on 6.0.14	24	20
OSSbsd	4.0, 4.1 (Jan.–Jul. 2007), 4.2 (Oct.–Dec. 2007)	Train on 4.0, 4.1 Test on 4.2	28	12
OSSmoz	1.0, 1.5 (Jul.–Dec. 2005), 2.0 (Jan.–Jun. 2006)	Train on 1.0, 1.5 Test on 2.0	72	24
IND01	4.3.0, 4.3.1, 4.4.0, 4.4.1, 4.5.0 (Oct. 2006–Feb. 2007), 4.5.1 (Mar.–Apr. 2007)	Train on 4.3.0, 4.3.1, 4.4.0, 4.4.1, 4.5.0 Test on 4.5.1	20	8
IND02	5.07, 5.09 (Feb. 2006–Apr. 2007), 5.10 (Feb.–Dec. 2007)	Train on 5.07, 5.09 Test on 5.10	38	11
IND03	5.09, 5.10 (Sept. 2005–Dec. 2007)	Train on 5.09 Test on 5.10	19	11
IND04	3.0, 3.1 (Jan. 2007–Mar. 2008), 3.2 (Sept.–Dec. 2008)	Train on 3.0, 3.1 Test on 3.2	60	16

4.4 Research questions

Before presenting the empirical study in detail, we pose the specific research questions to be answered. Informally, we want to evaluate if there can be a better approach for cross-release prediction of fault count data in general when comparing traditional and machine learning approaches. We quantify this evaluation in terms of goodness of fit, predictive accuracy, model bias and qualitative criteria:

RQ 1: What is the *goodness of fit (gof)* of traditional and machine learning models for cross-release fault count predictions?

RQ 2: What are the levels of *predictive accuracy* of traditional and machine learning models for cross-release fault count predictions?

RQ 3: What is the *prediction bias* of traditional and machine learning models for cross-release fault count predictions?

RQ 4: How do the prediction techniques compare *qualitatively* in terms of generality, transparency, configurability and complexity?

4.5 Evaluation measures

Selecting appropriate evaluation measures for comparing the predictability of competing models is not trivial. A number of different accuracy indicators have been used for comparative analysis of models, see e.g., [289]. Since a comparison of different measures is out of scope for this chapter, we used multiple evaluation measures to increase confidence in model predictions; a recommended approach since we would have a hard time relying on a single evaluation measure [257].

However, quantitative evaluations of predictive accuracy and bias are not the only important aspects for practical use of the modeling techniques. Hence, we also compare them on a set of qualitative aspects. Below we describe both of these types of evaluation.

4.5.1 Quantitative evaluation

On the quantitative front, we test the models' results for goodness of fit, predictive accuracy and model bias. A goodness of fit test measures the difference between the observed and the fitted values after a model is fitted to the training data. We are interested here to test whether the two samples (actual fault count data from the testing set and the predicted fault count data from each technique) belong to identical distributions. Therefore, the Kolmogorov-Smirnov (K-S) test is applied which is a commonly used statistical test for measuring goodness of fit [305, 231]. The K-S test is distribution free, which suited the samples as they failed the normality tests. Since goodness of fit tests do not measure predictive accuracy *per se*, we use prequential likelihood ratio (PLR), absolute average error (AAE), absolute relative error (ARE) and prediction at level l, pred(l), as the measures for evaluating predictive accuracy. Specifically, PLR provides a measure for short-term predictability (or next-step predictability) while AAE and ARE provides measures for variable-term predictability [181, 229]. We further test a particular model's bias which gives an indication of whether the model is prone to overestimation or underestimation [229]. To measure a particular model's bias, we examine the distribution of residuals to compare models as suggested in [193, 270]. We also formally test for significant differences between competing prediction systems as recommended in e.g., [289]. In the following we describe the evaluation measures in more detail.

Kolmogorov-Smirnov (K-S) test The details regarding this test are given in Section 3.5.1 of the thesis.

Prequential likelihood ratio (PLR) PLR is used to investigate the relative plausibility of the predictions from two models [1]. The prequential likelihood (PL) is the measure of closeness of a model's probability density function to the true probability density function. It is defined as the running product of one-step ahead predictions $\hat{f}_i(t_i)$ of next fault count intervals $T_{j+1}, T_{j+2}, \dots, T_{j+n}$,

$$PL_n = \prod_{i=j+1}^{j+n} \hat{f}_i(t_i)$$

The PLR of two prediction systems, A and B, is then the running product of the ratio of their successive one-step ahead predictions $\hat{f}_j^A(t_j)$ and $\hat{f}_j^B(t_j)$ respectively [51]:

$$\mathrm{PLR}_i^{AB} = \prod_{j=s}^{j=i} \frac{\hat{f}_j^A(t_j)}{\hat{f}_j^B(t_j)}$$

In our case, we select the actual time distribution of fault count data as a reference and conduct pair-wise comparisons of all models' predictions against it. Then the model with the relatively smallest prequential likelihood ratio can be expected to provide the most trustworthiest predictions. For further details on PLR, see [51].

Absolute average error (AAE) and relative error (ARE) The AAE is given by,

$$\mathrm{AAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

where $\hat{y}_i$ is the predicted value against the original y_i, n is the total number of points in the test data set.

The ARE is given by,

$$\text{ARE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_i|}$$

where $\hat{y}_i$ is the predicted value against the original y_i, n is the total number of points in the test data set.

Prediction at level l Prediction at level l, $\text{pred}(l)$, represents a measure of the number of predictions within l% of the actuals. We have used the standard criterion for considering a model as acceptable which is $\text{pred}(0.25) \geq 0.75$ which means that at least 75% of the estimates are within the range of 25% of the actual values [87].

Distribution of residuals To measure a particular model bias, we examine the distribution of residuals to compare models [193, 289]. It has the convenience of applying significance tests and visualizing differences in absolute residuals of competing models using box plots.

4.5.2 Qualitative evaluation

In addition to the quantitative evaluation factors there are other qualitative criteria, which needs to be accounted for when assessing the usefulness of a particular modeling technique. Qualitative criterion-based evaluation evaluates each method based on conceptual requirements [113]. One or more of these requirements might influence model selection. We use the following qualitative criteria [56, 113, 225, 227], which we believe are important factors influencing model selection:

1. Configurability (ease of configuration), i.e., how easy is it to configure the technique used for modeling?

2. Transparency of the solution (explanatory value regarding output), i.e., do the models explain the output?

3. Generality (applicability in varying operational environments), i.e., what is the extent of generality of model results for diverse data sets?

4. Complexity, i.e., how complex are the resulting models?

4.6 Software fault prediction techniques

This section describes the techniques used in this study for software fault prediction. The techniques include genetic programming (GP), artificial neural networks (ANN), support vector machine regression (SVM), Goel-Okumoto non-homogeneous Poisson process model (GO), Yamada's *S*-shaped growth model (YAM) and Brooks and Motley's Poisson model (BMP). We have used GPLAB version 3 [292] (for running GP), WEKA software version 3.4.13 [341] (for running ANN, SVM and LR) and SMERFS3 version 2 [97] (for running GO, YAM and BMP).

4.6.1 Genetic programming (GP)

The background to GP is given in Section 3.3 of this thesis. Below we discuss the parameter settings for GP.

Initially we experimented with a minimal set of functions and the terminal set containing the independent variable only. We incrementally increased the function set with additional functions and later on also complemented the terminal set with a random constant. For each data set, the best model having the best fitness was chosen from all the runs of the GP system with different variations of function and terminal sets. The GP programs were evaluated according to the sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$, where e_i is the actual fault count data, $e_i^{'}$ is the estimated value of the fault count data and n is the size of the data set used to train the GP models. The control parameters that were chosen for the GP system are shown in Table 4.2. The selection method used is $lexictour$ in which the best individuals are selected from a random number of individuals. If two individuals are equally fit, the tree with fewer nodes is chosen as the best [292]. For a new population, the parents and offsprings are prioritized for survival according to elitism. The elitism level specifies the members of the new population, to be selected from the current population and the newly generated individuals. The elitism level used in this study is $replace$ in which children replace the parent population having received higher priority of survival, even if they are worse than their parents [292].

4.6.2 Artificial neural networks (ANN)

The development of artificial neural networks is inspired by the interconnections of biological neurons [286]. These neurons, also called nodes or units, are connected by direct links. These links are associated with numeric weights which shows both the strength and sign of the connection [286]. Each neuron computes the weighted sum

Table 4.2: GP control parameters.

Control parameter	Value
Population size	200
Number of generations	450
Termination condition	450 generations
Function set (for OSStom, OSSbsd, IND01 & IND02)	$\{+,-,*,\sin,\cos,\log,\text{sqrt}\}$
Function set (for OSSmoz, IND03, IND04)	$\{+,-,*,/,\sin,\cos,\log\}$
Terminal set	$\{x\}$
Tree initialization (for OSStom, OSSbsd, OSSmoz, IND03, IND04)	Ramped half-and-half method
Tree initialization (for IND01, IND02)	Full method
Genetic operators	Crossover, mutation, reproduction
Selection method	Lexictour
Elitism	Replace

of its input, applies an activation (step or transfer) function to this sum and generates output, which is passed on to other neurons.

A neural network structure can be feed-forward (acyclic) network and recurrent (cyclic) network. Feed-forward neural networks do not contain any cycles and a network's output is only dependent on the current input instance [341]. Recurrent neural networks feeds its output back into it's own inputs, supporting short-term memory. Feed-forward neural network are more common and may consist of three layers: Input, hidden and output. The feed-forward neural network having one or more hidden layers is called multilayer feed-forward neural network. Back-propagation is the common method used for learning the multilayer feed-forward neural network whereby the error from the output layer back-propagates to the hidden layer. The ANN models for this study were obtained using multilayer feed-forward neural networks containing one input layer, one hidden layer and one output layer. The default parameter values for multilayer perceptron implemented in Weka software version 3.4.13 were used for training. The output layer had one node with linear transfer function and the two nodes in the hidden layer had sigmoid transfer function.

4.6.3 Support vector machine (SVM)

Support vector regression uses a support vector machine algorithm for numeric prediction. Support vector machine algorithms classify data points by finding an optimal linear separator which possess the largest margin between it and the one set of data points on one side and the other set of examples on the other. The largest separator is found by solving a quadratic programming optimization problem. The data points closest to the separator are called support vectors [286]. For regression, the basic idea is to discard the deviations up to a user specified parameter $\in$ [341]. Apart from specifying $\in$, the upper limit C on the absolute value of the weights associated with each data point has to be enforced (known as capacity control). The default parameter values for support vector regression implemented in Weka software version 3.4.13 were used for training. More details on support vector regression can be found in [115].

4.6.4 Linear regression (LR)

The linear regression used in the study performs a standard least-squares linear regression [159]. Simple linear regression helps to find a relationship between the independent (x) and dependent (y) variables. It also allows for prediction of dependent variable values given values of the independent variable.

4.6.5 Traditional software reliability growth models

As discussed in Section 4.1, we use three traditional software reliability growth models for comparisons. Below is a brief summary of these models while further details, regarding e.g., the models' assumptions, can be found in [52, 109, 347].

The Goel-Okumoto non-homogeneous Poisson process model (GO) [109] is given by,

$$m(t) = a[1 - e^{-bt}] \tag{4.1}$$

while Yamada's S-shaped growth model (YAM) [347] is also a non-homogeneous Poisson process model given by,

$$m(t) = a(1 - (1 + bt)e^{-bt}) \tag{4.2}$$

where in both above equations a is the expected total number of faults before testing, b is the failure detection rate and $m(t)$ is the expected number of faults detected by time t, also called as the mean value function. In the above two models, the failure arrival process is viewed as a stochastic non-homogeneous Poisson process (NHPP), with the number of failures $X(t)$ for a given time interval $(0,t)$ given by the probability $P[X(t) = n]$ as [309]:

$$P[X(t) = n] = \frac{[m(t)]^n e^{-m(t)}}{n!} \tag{4.3}$$

Brooks and Motley's model come in two variations, depending upon the assumption of either a Poisson or a binomial distribution of failure observations. We make use of the Poisson model (BMP) [52]. The BMP model, with a Poisson distribution of failure observations n_i over all possible X for i-th period, of length t_i, gives the probability $P[X = n_i]$ of number of failures for a given time interval,

$$P[X = n_i] = \begin{cases} \frac{(N_i \phi_i)^{n_i} e^{-N_i \phi_i}}{n_i!} \\ \phi_i = 1 - (1 - \phi)^{t_i} \end{cases} \tag{4.4}$$

where N_i is the estimated number of defects at the beginning of i-th period and ϕ is Poisson constant.

4.7 Experiment and results

We collected data from seven multi-release open source and industrial software projects for the purpose of cross-release prediction of fault count data. The data sets have been impartially split into training and test sets. The training set is used to build the models while the independent test set is used to evaluate the models' performance. The performance is assessed both quantitatively (goodness of fit, predictive accuracy, model bias) and qualitatively (ease of configuration, solution transparency, generality and complexity). The independent variable in our case is the week number while the corresponding dependent variable is the count of faults. Week number is taken as the independent variable because it is controllable and potentially has an effect on the dependent variable, i.e., the count of faults, in which the effect of the treatment is measured. The design type of our experiment is one factor with more than two treatments [40]. The factor is the prediction of fault count data while the treatments are the application of GP, traditional approaches and the machine learning approaches. In this section, we further present the results of goodness of fit, predictive accuracy, model bias and qualitative evaluation for different techniques applied to the different data sets in the study.

4.7.1 Evaluation of goodness of fit

We make use of K-S test statistic to test whether the two samples (in this case, the predicted and actual fault count data from the test set part of the data set for each technique) have the same probability distribution and hence represents the same population (more details regarding this test appear in Section 3.5.1 of this thesis).

Table 4.3 shows the results of applying K-S test statistic for each technique for every data set. The (–) in the Table 4.3 indicates that the algorithm was not able to

Table 4.3: Results of applying Kolmogorov-Smirnov test. The bold values indicate $J < J_{\alpha}$, (–) indicates lack of model convergence, J_{α} is the critical J value at α=0.05

	Sample size	J_{GP}	J_{ANN}	J_{SVM}	J_{LR}	J_{GO}	J_{YAM}	J_{BMP}	$J_{\alpha=0.05}$
OSStom	20	**0.20**	0.95	**0.30**	**0.25**	–	**0.25**	**0.25**	0.43
OSSbsd	12	**0.17**	**0.50**	0.75	**0.50**	**0.42**	**0.58**	**0.50**	0.68
OSSmoz	24	0.46	**0.37**	1.00	1.00	–	**0.17**	0.46	0.39
IND01	8	**0.37**	0.87	1.00	1.00	–	0.75	**0.62**	0.75
IND02	11	**0.27**	**0.45**	**0.27**	**0.27**	**0.27**	**0.54**	**0.27**	0.64
IND03	11	**0.54**	**0.54**	–	**0.54**	–	–	0.82	0.64
IND04	16	0.50	1.00	1.00	1.00	–	1.00	1.00	0.48

converge for the particular data set. The instances where the K-S statistic J is less than the critical value J_{α} are shown in bold in Table 4.3. It is evident from Table 4.3 that GP was able to show statistically significant goodness of fit for the maximum number of data sets (i.e., five). The other close competitors were ANN (4), LR (4), YAM (4) and BMP (4). This indicates that, at significance level $\alpha = 0.05$, GP is better in terms of having statistically significant goodness of fit on more data sets than other, competing, techniques.

Table 4.4 summarizes the K-S test statistic for all the techniques. Since some techniques did not converge for some data sets, the number of data sets applicable for techniques is different. GP, ANN, LR and BMP were able to converge for all seven data sets. However, the same did not happen with other techniques, as can be seen from the second column of Table 4.4. We can observe that in comparison with ANN, LR and BMP, with seven data sets each, GP appears to be a better technique (showing a comparatively closer fit to the set of observations) when ranked based on mean and median.

Table 4.4: Summary statistics for K-S test showing the mean, median, min and max corresponding to the respective number of data sets.

		K-S test statistic			
Technique	No. of data sets	Mean	Median	Min	Max
GP	7	0.36	0.37	0.17	0.54
BMP	7	0.56	0.50	0.25	1.00
LR	7	0.65	0.54	0.25	1.00
ANN	7	0.67	0.54	0.37	1.00
YAM	6	0.55	0.56	0.17	1.00
SVM	6	0.72	0.87	0.27	1.00
GO	2	0.34	0.34	0.27	0.42

We conclude that the goodness of fit of GP models for cross-release predictions is promising in comparison with traditional and machine learning models as they were able to show better goodness of fit for majority of the data sets, both in terms of K-S test statistic and ranking based on mean and median, on more data sets.

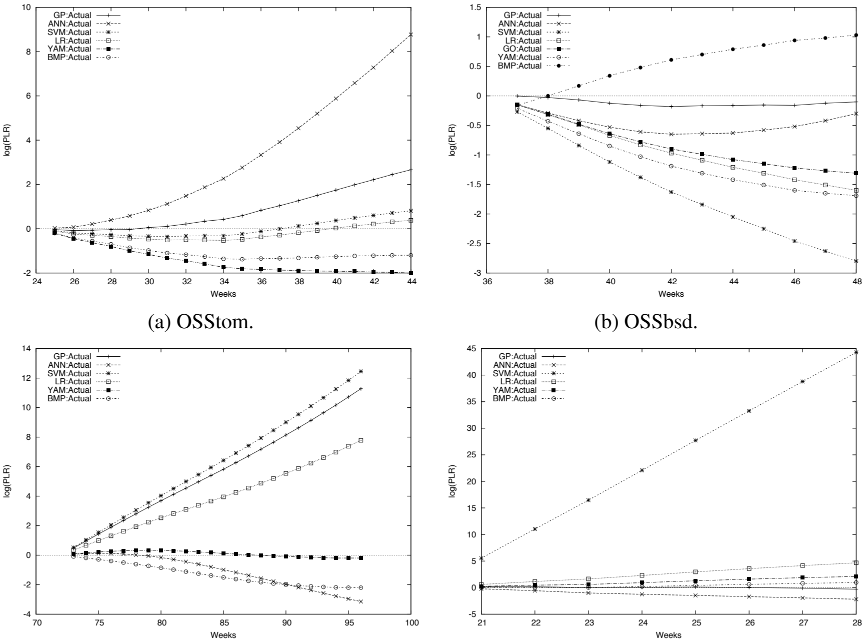
4.7.2 Evaluation of predictive accuracy

Table 4.5 shows the final *log* result of the running product of the ratio of the successive one-step ahead predictions of actual fault count data and other techniques' prediction. Since the actual time distribution of weekly/monthly fault count data is chosen as the reference, the PLR values closer to 0 are better. We can observe, from Table 4.5, that the $log(\text{PLR})$ values are closest to 0 on four occasions for GP while thrice for LR. The 'winner' from each data set is shown in bold in Table 4.5. This shows that for most data sets (four out of seven), the probability density function of the GP model is closer to the true probability density function.

Table 4.5: $log(\text{PLR})$ values for one-step-ahead predictions. The values shown are the final *log* result of the running product of ratio of the successive on-step ahead predictions of actual fault count and other models' predictions. (Values closer to 0 are of course better.)

	Sample size	GP	ANN	SVM	LR	GO	YAM	BMP
OSStom	20	2.66	8.77	0.81	**0.38**	–	−2.00	−1.20
OSSbsd	12	**−0.10**	−0.30	−2.80	−1.60	−1.31	−1.69	1.03
OSSmoz	24	11.28	−3.14	12.45	7.78	–	**−0.19**	−2.20
IND01	8	**−0.29**	−2.17	44.28	4.67	–	2.11	0.97
IND02	11	0.15	0.74	0.39	**0.07**	−0.55	−0.88	0.56
IND03	11	**7.21**	**7.21**	–	**7.21**	–	–	−8.62
IND04	16	**0.56**	1.14	−6.63	−6.75	–	−7.58	−7.17

Figures 4.2a – 4.2g depicts the PLR analysis for all the data sets which shows the pair-wise comparisons of each technique with the actual weekly/monthly fault count data which has been chosen as the reference model (indicated as a dotted straight line in the plots of Figures 4.2a – 4.2g). We see that for OSStom (Figure 4.2a), the prediction curves for LR and SVM are closer to the reference in comparison with other curves. For OSSbsd (Figure 4.2b), the prediction curve for GP follows the reference more closely than other curves. The same behavior is also evident for IND01, IND03 and IND04 (Figures 4.2d, 4.2e and 4.2g). However, for OSSmoz (Figure 4.2c), YAM is better at following the reference compared to any other curve, while for IND03 (Figure 4.2f), the curves for GP, ANN and LR are much closer to the $log(\text{PLR})$ of actual fault count data. Overall, GP was able to show more consistent predictive accuracy, across four



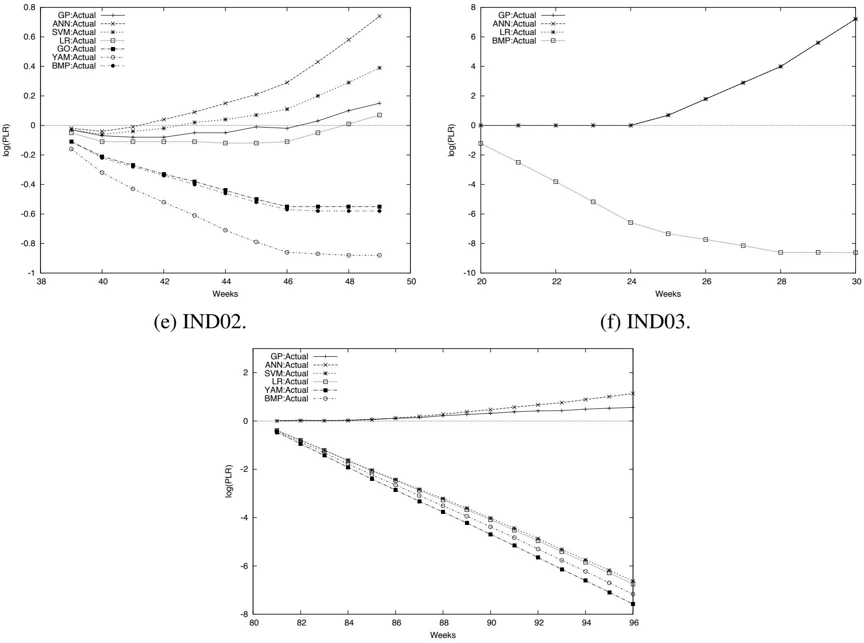
(a) OSStom.

(b) OSSbsd.

(c) OSSmoz.

(d) IND01.

Figure 4.2: log(PLR) plots for the data sets OSStom, OSSbsd, OSSmoz, IND01, IND02, IND03 and IND04. Continuing on to the next page.



(e) IND02.

(f) IND03.

(g) IND04.

Figure 4.2: Continuing from the previous page; $log(\text{PLR})$ plots for the data sets OSStom, OSSbsd, OSSmoz, IND01, IND02, IND03 and IND04.

of the seven data sets. Table 4.6 shows the computed values of AAE for all the data sets. The lowest AAE values from each data set are shown in bold. GP gave the lowest AAE values for the maximum number of data sets (data sets OSStom, OSSbsd, IND01, IND03 and IND04) followed by LR, which remained successful in case of data sets IND02 and IND03.

Table 4.6: AAE values for different techniques for all data sets. The bold values indicate the lowest AAE values from each data set. (–) indicates lack of model convergence.

	Sample size	GP	ANN	SVM	LR	GO	YAM	BMP
OSStom	20	**6.35**	35.38	7.55	6.91	–	8.36	**6.35**
OSSbsd	12	**3.78**	14.08	44.01	23.72	18.93	24.79	19.44
OSSmoz	24	64.71	45.18	114.69	78.61	–	**9.77**	26.12
IND01	8	**2.90**	8.49	27.64	12.29	–	6.51	3.47
IND02	11	5.07	12.05	7.57	**4.58**	7.80	12.60	8.25
IND03	11	**1.36**	**1.36**	–	**1.36**	–	–	1.90
IND04	16	**1.18**	2.31	17.08	17.46	–	20.12	18.80

Since the AAE samples from different methods did not satisfy the normality assumption, we used the non-parametric Wilcoxon rank sum test to test the null hypothesis that data from two samples have equal means. We tested the following pairs of AAE samples: GP vs. ANN, GP vs. SVM, GP vs. LR and GP vs. YAM. The corresponding p-values for these tests came out to be 0.27, 0.02, 0.13 and 0.03 respectively. At significance level of 0.05, the results indicate that the null hypothesis can be rejected for GP vs. SVM and GP vs. YAM, while, on the other hand, there is no statistically significant difference between the AAE means of GP, ANN and LR at the 0.05 significance level.

Apart from statistical testing, Table 4.7 presents the summary statistics of AAE for all the techniques.

We can observe that having a ranking based on median, GP has the lowest value in comparison with ANN, LR and BMP having seven data sets each. For a ranking based on mean, GP appears to be very close to the best mean AAE value for BMP, which is 12.05.

Table 4.8 shows the computed values of ARE for all the data sets. It is evident from the table that GP resulted in the lowest ARE values for most of the data sets (five out of seven). The other closest technique was LR that was able to produce lowest ARE values for two data sets. This indicates that GP is generally a better approach for variable-term predictability.

As with AAE, ARE samples from different methods also did not satisfy the nor-

Table 4.7: Summary statistics for AAE showing the mean, median, min and max corresponding to the respective number of data sets.

		AAE statistic			
Technique	No. of data sets	Mean	Median	Min	Max
BMP	7	12.05	8.25	1.90	26.12
GP	7	12.19	3.78	1.18	64.71
ANN	7	16.98	12.05	1.36	45.18
LR	7	20.70	12.29	1.36	78.61
YAM	6	13.69	11.18	6.51	24.79
SVM	6	36.42	22.36	7.55	114.69
GO	2	13.36	13.36	7.80	18.93

Table 4.8: ARE values for different techniques for all data sets. Bold values indicate the lowest ARE values from each data set. (–) indicates lack of model convergence.

	Sample size	GP	ANN	SVM	LR	GO	YAM	BMP
OSStom	20	**0.06**	0.33	0.07	0.07	–	0.11	0.08
OSSbsd	12	**0.02**	0.08	0.26	0.14	0.12	0.15	0.12
OSSmoz	24	0.22	0.15	0.40	0.28	–	**0.03**	0.10
IND01	8	**0.10**	0.31	1.00	0.44	–	0.23	0.11
IND02	11	0.03	0.07	0.04	**0.02**	0.05	0.08	0.05
IND03	11	**0.37**	**0.37**	–	**0.37**	–	–	1.49
IND04	16	**0.03**	0.07	0.51	0.52	–	0.61	0.57

mality assumption. We used the non-parametric Wilcoxon rank sum test for testing the following pairs of ARE samples: GP vs. ANN, GP vs. SVM, GP vs. LR and GP vs. YAM. The corresponding p-values for these tests came out to be 0.18, 0.07, 0.15 and 0.29 respectively. This shows that, for significance level of 0.05, there is no statistical difference between the ARE means of GP, ANN, SVM, LR and YAM.

Apart from statistical testing, we can observe from Table 4.9 that having a ranking based on both mean and median; GP has the lowest value in comparison with ANN, LR and BMP having seven data sets each.

We further applied the measure of pred(l) to judge on the predictive ability of the prediction systems. The result of applying pred(l) is shown in Table 4.10.

The standard criterion of pred$(0.25) \geq 75$ for stable model predictions was met by different techniques for different data sets, but GP and BMP were able to meet this criterion on most data sets i.e., five. The application of these two techniques on the five data sets resulted in having 100% of the estimates within the range of 25% of the actual values.

We conclude that while the statistical tests for AAE and ARE do not give us a clear indication of a particular technique being (statistically) significantly better compared

Table 4.9: Summary statistics for ARE showing the mean, median, min and max corresponding to the respective number of data sets.

		ARE statistic			
Technique	No. of data sets	Mean	Median	Min	Max
GP	7	0.12	0.06	0.02	0.37
ANN	7	0.20	0.15	0.07	0.37
LR	7	0.26	0.28	0.02	0.52
BMP	7	0.26	0.11	0.05	0.80
YAM	6	0.20	0.13	0.03	0.61
SVM	6	0.37	0.33	0.04	0.96
GO	2	0.08	0.08	0.05	0.12

Table 4.10: pred(0.25) calculation for different techniques for all data sets. (–) shows lack of model convergence.

	Sample size	GP (%)	ANN (%)	SVM (%)	LR (%)	GO (%)	YAM (%)	BMP (%)
OSStom	20	100	30	100	100	–	99	100
OSSbsd	12	100	100	50	100	100	100	100
OSSmoz	24	41.67	100	0	16.67	–	100	100
IND01	8	100	37.5	100	0	–	37.5	100
IND02	11	100	45.45	100	100	100	100	100
IND03	11	45.45	45.45	–	44.45	–	–	18.18
IND04	16	100	100	0	0	–	0	0

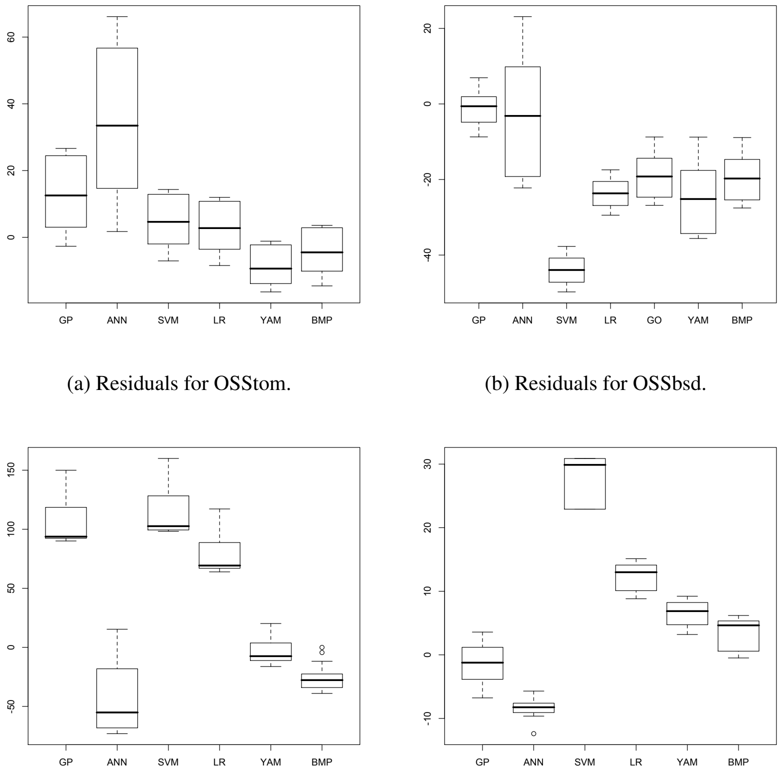
to other techniques, the summary statistics (Tables 4.7 and 4.9) together with the evaluation of pred(0.25) and PLR show that the use of GP for cross-release prediction of fault count data is in many ways better in comparison with other techniques.

4.7.3 Evaluation of model bias

We examined the bias in predictions by making use of box plots of model residuals. The box plots of residuals for all the data sets are shown in Figures 4.3a – 4.3g. For OSSbsd (Figure 4.3b) and IND04 (Figure 4.3g), the box plots for GP show two important characteristics:

1. Smaller or equivalent length of the box plot as compared with other box plots.
2. Presence of majority of the residuals close to 0 as compared with other box plots.

For IND03 (Figure 4.3f), the length of the box plot and its proximity close to 0 appear to be similar for GP, ANN and LR. For OSStom (Figure 4.3a), SVM and LR are better placed than the rest of the techniques while for OSSmoz (Figure 4.3c), YAM



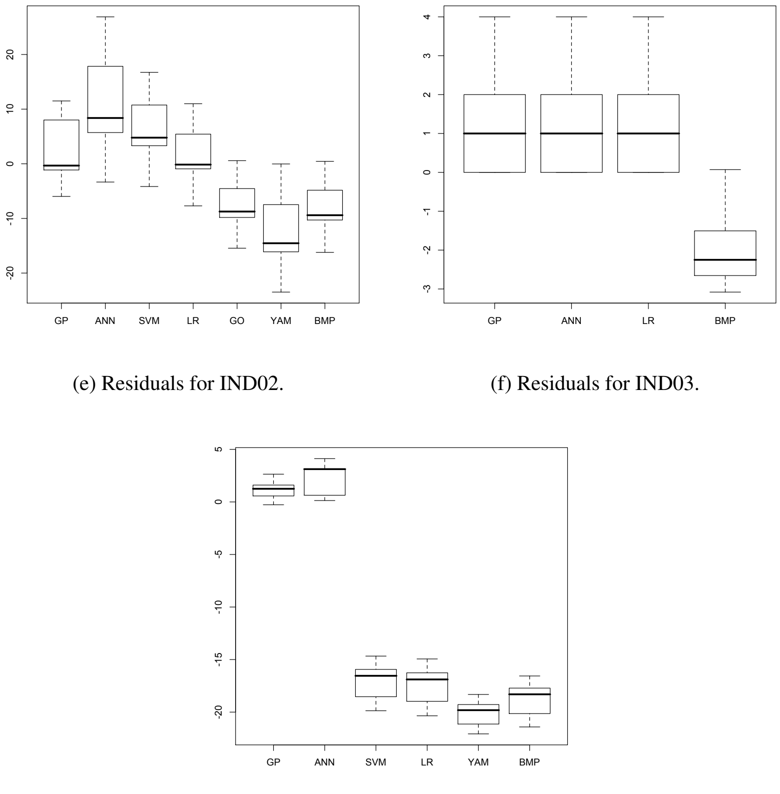
(a) Residuals for OSStom.

(b) Residuals for OSSbsd.

(c) Residuals for OSSmoz.

(d) Residuals for IND01.

Figure 4.3: Charts showing box plots of residuals for the seven data sets. Continuing on to the next page.



(e) Residuals for IND02.

(f) Residuals for IND03.

(g) Residuals for IND04.

Figure 4.3: Continuing from the previous page; Charts showing box plots of residuals for the seven data sets.

appears to be having a smaller box plot positioned in the proximity of 0. For IND01, although the length of the box plot seems to be small for ANN, it still appears below the 0-mark indicating that the predictions from ANN are overestimating the actual fault count data. The GP box plot, however, appears to be better positioned in this respect. The same is the case with IND02 (Figure 4.3e) where GP and LR show a good trade-off between length and actual position of the box plot.

Since the box plots of the residuals were skewed, we resorted to using the non-parametric Kruskal-Wallis test to examine if there is a statistical difference between the residuals for all the data sets and to confirm the trend observed from the box plots. The results of the application of the Kruskal-Wallis test appear in Table 4.11. For each of the data sets, the Kruskal-Wallis statistic h is greater than the critical value $\chi^2_{0.05}$. Hence, we have sufficient evidence to reject the null hypothesis that the residuals for different techniques within a project are similar.

Table 4.11: Kruskal-Wallis statistic h for different data sets for testing difference in residuals. ν is the degrees of freedom.

Data sets	Kruskal-Wallis statistic, h
OSStom, $\chi^2_{0.05}$=11.07, $\nu = 5$	83.49
OSSbsd, $\chi^2_{0.05}$=12.60, $\nu = 6$	58.51
OSSmoz, $\chi^2_{0.05}$=11.07, $\nu = 5$	122.95
IND01, $\chi^2_{0.05}$=11.07, $\nu = 5$	43.76
IND02, $\chi^2_{0.05}$=12.60, $\nu = 6$	42.9
IND03, $\chi^2_{0.05}$=7.81, $\nu = 3$	21.45
IND04, $\chi^2_{0.05}$=11.07, $\nu = 5$	75.57

In order to further investigate if the residuals obtained from GP are different from those of other techniques, we used the Wilcoxon rank sum test. The p-values obtained are shown in Table 4.12. The table shows that, except for four cases, the p-values were found to be less than 0.05 thus rejecting the null hypothesis that the samples are drawn from identical continuous distributions. The four cases where the null hypothesis was not rejected coincide with data sets OSSbsd and IND02, where the comparisons of the residuals of GP were not found to be different from those of ANN, SVM and LR. (These cases are shown in bold in Table 4.12.) We conclude that in terms of model bias, the examination of residuals show the greater consistency of GP, as compared with other traditional and machine learning models (in having predictions that result in smaller box plots that are positioned near the 0-mark). Further, application of the Wilcoxon rank sum test shows that except for four combinations (GP:ANN-OSSbsd, GP:SVM-IND02, GP:LR-IND02, GP:ANN-IND02), there is sufficient evidence to show that the residuals from GP are different from those of other competing

Table 4.12: p-values after applying the Wilcoxon rank sum test on residuals (values rounded to two decimal places). Values in bold indicate $p > 0.05$.

	$P_{GP:ANN}$	$P_{GP:SVM}$	$P_{GP:LR}$	$P_{GP:GO}$	$P_{GP:YAM}$	$P_{GP:BMP}$
OSStom, $\alpha = 0.05$	0.00	0.00	0.01	–	0.00	0.00
OSSbsd, $\alpha = 0.05$	**0.79**	0.00	0.00	0.00	0.00	0.00
OSSmoz, $\alpha = 0.05$	0.00	0.02	0.00	–	0.00	0.00
IND01, $\alpha = 0.05$	0.00	0.00	0.00	–	0.00	0.02
IND02, $\alpha = 0.05$	**0.09**	**0.32**	**0.95**	0.00	0.00	0.00
IND03, $\alpha = 0.05$	–	–	–	–	–	0.00
IND04, $\alpha = 0.05$	0.02	0.00	0.00	–	0.00	0.00

techniques.

4.7.4 Qualitative evaluation of models

The selection of a particular model for fault count predictions is influenced not only by the quantitative factors (e.g., goodness of fit, predictive accuracy and bias) but also by certain conceptual requirements, which we term as qualitative measures. We believe that it is important to take into account these qualitative measures (in addition to quantitative ones) to reach an informed decision about a suitable technique or combination of techniques to use for fault count predictions.

Ease of configuration The parametric models including BMP, GO, YAM and linear regression require an estimation of certain parameters. The number of these parameters, and the ease with which these parameters can be measured, affects measurement cost [225]. With automated reliability measurement using tools such as CASRE (Computer-Aided Software Reliability Estimation) and SMERFS (Statistical Modeling and Estimation of Reliability Functions for Systems) [97, 256], the estimation of parameters may have eased but such tools are limited by the number of supported models and numerical approximation methods. Linear regression, in comparison, is much simpler to use having several tools available for automation.

For the machine learning methods, as used in this study, the ease of configuration concerns setting algorithmic control parameters. For ANN, some initial experimentation is required to reach a suitable configuration of number of layers and associated number of neurons. For GP, there are several parameters that control the adaptive evaluation of fitter solutions, such as selection of function and terminal sets and probabilities of genetic operators. For SVM, one needs to take care of capacity control and the loss function. But once these algorithmic control parameters are set, an approximation is found by these methods during training. However, there seems to be no clear differentiation among different techniques with respect to ease of configuration. This is in our opinion a general problem and indicates a need for further research.

Transparency of the solution The resulting equations for traditional models are partially transparent; however, GP is capable of producing transparent solutions because the resulting model is an algebraic expression (which is not the case with ANN and SVM). Thus, transparency of solutions is one distinct advantage of using GP. Transparency of the solutions *can be important for the purpose of verification as well as theory building and gaining an understanding of the process being modeled* [113]. In our case, with one independent variable (week number) and one dependent variable (count of faults), typical GP solutions are of the form below:

$$times(minus(sin(minus(cos(x),x)),minus(log(cos(log(sin(log(x))))),sin(x))),log(x))$$

where x is the independent variable and *minus*, *times*, *sin*, *cos*, *log* represents the function set (as outlined in Table 4.2).

Generality The extent of generality of model results for diverse data sets is better for machine learning and evolutionary methods than the traditional methods. This is because of the fact that machine learning and evolutionary models do not depend on prior assumptions about data distribution and form of relationship between independent and dependent variables. The model and the associated coefficients are evolved based on the fault data collected during the initial test phase. In this sense, the applicability of the models derived from machine learning and evolutionary methods for different development and operational environments and life-cycle phases, appear to be better suited than traditional modeling techniques.

Complexity The complexity criterion is especially important to discuss with respect to GP since GP has the potential of evolving transparent solutions. However the solutions can become complex as the number of nodes in the GP solution increases, a phenomenon known as bloating. Although there are different ways to control this (see e.g., [222]), in the context of canonical GP, this is still an important consideration. For ANN the complexity can be connected to the potential complex and inefficient structures, which can evolve in an attempt to discover difficult data patterns. For SVM and traditional software reliability growth models, being essentially black-box, the complexity is difficult to discuss. However, for linear regression, where the reasoning process is partially visible, the complexity is apparently minimal.

There can be another way to evaluate complexity in terms of suitability of a technique to incorporate complex models. This can be connected back to the theory of whether the modeling technique determines its own structure or requires the engineer to provide the structure of the relationship between independent and dependent variables [113]. The machine learning and evolutionary models certainly scores high in this respect in comparison with traditional methods.

4.8 Empirical validity evaluation

There can be different threats to the validity of the empirical results [342]. In this section we cover, conclusion, internal, construct and external validity threats. *Conclusion validity* refers to the statistically significant relationship between the treatment and outcome. We have used non-parametric statistics in this study, particularly Kolmogorov-Smirnov goodness of fit test, Kruskal-Wallis statistic and Wilcoxon rank sum test. Although the power of parametric tests is known to be higher than for non-parametric tests, we were uncertain about the corresponding parametric alternatives meeting the tests' assumptions. Secondly, we used a significance level of 0.05, which is a commonly used significance level for hypothesis testing [158]; however, facing some criticism lately [142]. Therefore, it can be considered as a limitation of our study and a potential threat to conclusion validity. One potential threat to conclusion validity could have been that the fitness evaluation used for GP (Subsection 4.6.1) is similar to the quantitative evaluation measures for comparing different techniques (Subsection 4.5.1). This is, however, not the case with this study since the GP fitness function differs from the quantitative evaluation measures and since we have used a variety of different quantitative evaluation measures not necessarily based on minimization of standard error. A potential threat to conclusion validity is that the fault count data sets did not consider the severity level of faults, rather treated all faults equally. This is a limitation of our study, and we acknowledge that by considering severity levels the conclusion validity of the study would have improved; but at the same time we are also apprehensive that subjective bias might result in wrong assignment of severity levels. Another potential threat to conclusion validity is the different lengths of training and test data sets, depending upon the fault counts from respective. We plan to investigate this in the future. *Internal validity* refers to a causal relationship between treatment (independent variable) and outcome (dependent variable). It concerns all the factors that are required for a well-designed study. As for the selection of different data sets, we opted for having data sets from varying domains. Moreover, for each data set, we used a consistent scheme of impartially splitting the data set into testing and training sets for all the techniques. A possible threat to internal validity is that we cannot publicize our industrial data sets due to proprietary concerns; therefore other researchers cannot make use of these data sets. However, we encourage other researchers to emulate our results using other publicly available data sets. The best we can do is to clearly state our research design and apply recommended approaches like statistical hypothesis testing to minimize the chances of unknown bias. Additionally, we have data included in this study that is freely available since it was collected from open source software.

Also, another threat is that the different techniques were applied over different data sets in approximate standard parameter settings. For the GP algorithm there are no standard setting for the function and terminal sets so we had to test a few different ones, while keeping other parameters constant, until some search success was seen. Even though this is standard practice when using GP systems, a potential threat is that it could bias the results.

The used data sets were grouped on a weekly or monthly basis. While some studies (e.g., [343]) have indicated that the grouping of data is not a threat, it is possible that more detailed and frequent date and time resolution, and thus prediction intervals, could affect the applicability of different modeling techniques. For example, linear regression models might have a relative advantage concerning data that is more regular, with less frequent changes. However, it is hard to predict such effects and without further study we cannot determine if it is really a threat. *Construct validity* is concerned with the relationship between theory and application. We attempted to present both quantitative and qualitative evaluation factors in the study for defining the different constructs. There is a threat that we might have missed one or more evaluation criteria, however the evaluation measures used in the study reflect the ones commonly used for evaluating prediction models. *External validity* is concerned with generalization of results outside the scope of the study. We used data sets from both open source and industrial software projects, which we believe adds to the generalizability of the study. Also the data sets cannot be regarded as toy problems as each one of them represented fault data from multiple software releases in industry and in open source projects. One threat to external validity is the selection of machine learning algorithms for comparison. Being a large field of research, new data mining algorithms are continuously being proposed. We used a small subset of the machine learning algorithms but we are confident that our subset is a fairly representative one, being based on techniques which have different modeling mechanism and are currently being actively researched.

4.9 Discussion and conclusions

In this chapter, we compared cross-release predictions of fault count data from models constructed using common machine learning and traditional techniques. The comparisons were based on measures of goodness of fit, predictive accuracy and model bias. We also presented an analysis of some of the conceptual requirements for a successful model (including ease of configuration, transparency of solution, generality and complexity). These conceptual requirements are important when considering the applicability of a prediction system [227] and should be taken into account along with the quantitative performance.

The quantitative results of comparing different techniques have shown some indication that GP can be one of the competitive techniques for software fault prediction. In terms of conceptual requirements, though ease of configuration might not be the favorable aspect of GP models, the transparency of solution and generality are factors that add further value to the quantitative potential of GP-evolved models.

The fact that no prior assumptions have to be made in terms of actual model form is a distinct advantage of machine learning approaches over linear regression and traditional models. The traditional techniques need to satisfy the underlying assumptions, which means that there is no assurance that these techniques would converge to a solution. This does not happen with GP and ANN machine learning techniques. This shows that the machine learning techniques tend to be more *flexible* than their traditional counterparts. This flexibility also contributes to the greater *generalizability* of machine learning models in a greater variety of software projects. GP offers flexibility by adjusting a variety of functions to the data points; thereby both structure and complexity of the model evolve during subsequent generations.

Considering the different trade-offs among competing models, it appears crucial to define the *success criterion* for an empirical modeling effort. Such a definition of success would help exploit the unique capabilities of different modeling techniques. For instance, if success is defined in terms of having only accurate predictions without the need of examining the relationship among variables in the form of a function, then artificial neural networks (ANN) might be a worthy candidate for selection (being known as universal approximators), provided that the requisite levels of model accuracy are satisfied. But selecting ANN as a modeling technique would mean that we have to be aware of its potential drawbacks:

1. Less flexible as the neural nets cannot be manipulated once the learning phase finishes [90]. This means that neural networks require frequent re-training once specific process conditions change and hence adds to the maintenance overhead.

2. Black-box approach, thus disadvantageous for experts who want to have an understanding and potential manipulation of variable interactions.

3. Possibility of having inefficient and non-parsimonious[5] structures.

[5] The parsimony says that the model with the smallest number of parameters is usually the best.

4. Potentially poor generalizability outside the range of the training data [197].

In contrast, GP possesses certain unique characteristics considering the above issues. Symbolic regression using GP is flexible because of its ability to adjust a variety of functions to the data points and the models returned by symbolic regression are open for interpretation. This also helps to identify significant variables, which in the longer run could be used in subsequent modeling to increase the efficiency of the modeling effort [198]. Hence, this might also be useful for an easy integration in existing industrial work processes whereby only those variables could be used.

A brief summary of the relative performance of different techniques is presented in Table 4.13.

Table 4.13: Summary of the relative strengths of the methods on different criteria; techniques are ranked according to the relative performance for maximum number of times on all the data sets.

	GP	ANN	SVM	LR	GO	YAM	BMP
Goodness of fit	+ +	-	-	0	-	-	-
Accuracy	+ +	-	- -	0	- -	-	-
Bias	+ +	+	-	0	- -	-	- -
Ease of configuration	0	0	0	+	0	0	0
Transparency of solution	+	-	-	0	0	0	0
Generality	+	+	+	-	-	-	-
Complexity	0	0	0	+	0	0	0
Key: + + very good, + good, 0 average, - bad, - - very bad							

The performance indicators in Table 4.13 are given as to summarize the detailed evaluation done in the study based on several measures (Section 4.5). The indicators (+ +, +, 0, -, - -) for the quantitative measures of goodness of fit, accuracy and model bias represent the relative performance of different techniques for largest number of times on different data sets, e.g., GP is ranked (+ +) on accuracy because of performing comparatively better on accuracy measures for greater number of data sets. The indicators for the qualitative measures represent the relative merits of the techniques as discussed in Subsection 4.7.4. Table 4.13 shows that GP has the advantage of having better goodness of fit and accuracy as compared to other techniques, even though no special adaptions were made to the canonical GP algorithm taking into account the time series nature of the data (GP and ANN are expected to perform better for time series prediction if there is a possibility to save state information between different steps of prediction which can be used to identify trends in the input data; however, we wanted to compare the performance for standard algorithms and any enhancements to these techniques is not addressed in this study).

Table 4.13 shows that the GP models also exhibit less model bias. On the other hand, the ease of configuration and complexity are not necessarily stronger points for GP models. It is interesting to observe that ANN does not perform as well as expected in terms of goodness of fit and accuracy. Linear regression was able to show normal predictions in terms of goodness of fit and accuracy but scores higher on ease of configuration (however lacking generality due to the need of satisfying underlying assumptions). SVM and the traditional models (GO, YAM, BMP) appear to have similar advantages and disadvantages, with YAM showing a slightly improved quantitative performance, while SVM possesses better generality across different data sets.

The most encouraging result of this study shows the feasibility of using GP as a prediction tool across different releases of software. This indicates that the development team can use GP to make important decisions related to the quality of their deliverables. GP models also showed a decent ability to adapt to different time spans of releases (on the basis of the different lengths of the testing sets for different data sets), which is also a positive indicator. The study shows that GP is least affected by moderate differences in the release durations and can predict decently with variable time units into future. Additionally, having evaluated the performance on diverse data sets from different application domains, further points out the flexibility of GP, i.e., suiting a variety of data sets.

In short, the use of GP can lead to improved predictions with the additional capabilities of solution transparency and generality across varying operational environments. Secondly the GP technique used in this chapter followed a standard/canonical approach. Several adaptations to the GP algorithm (e.g., Pareto GP and grammar-guided GP) can potentially lead to further improved GP search process. We intend to investigate this in the future. Another future work involves evaluating the use of GP in an *on-going* project in an industrial context and compare the relative short-term and long-term predictive strength of the GP-evolved models for different lengths of training data.

The next Chapter 5 aims at predicting the faults-slip-through metric for test phase efficiency measurements. We also present the usefulness of such a prediction task and highlight criteria that are important in making the prediction techniques usable in industrial practice.

Chapter 5

Prediction of faults-slip-through in large software projects: An empirical evaluation

Originally published in Proceedings
of the 2nd International Symposium
on Search Based Software
Engineering (SSBSE'10), extended
version submitted to IEEE
Transactions on Software Engineering

W. Afzal, R. Torkar, R. Feldt,
T. Gorschek and G. Wikstrand

5.1 Introduction and problem statement

Presence of faults[1] usually indicates an absence of software quality. Software testing is the major fault-finding activity, therefore much research has focused on making the software test process as efficient and as effective as possible. One way to improve the test process efficiency is to avoid unnecessary rework by finding more faults earlier.

[1] According to IEEE Standard Glossary of Software Engineering Terminology [301], a fault is a manifestation of a human mistake.

This argument is based on the premise that the faults are cheaper to find and remove earlier in the software development process [36]. The faults-slip-through (FST) metric [79, 80] is one way of providing quantified decision-support to reduce the effort spent on rework.

The FST metric is used for determining whether a fault slipped through the phase where it should have been found or not [79, 80]. The term phase refers to any phase in a typical software development life cycle (as an example, ISO/IEC 12207 [298] defines the different software development phases). However, the most interesting and industry-supported applications of FST measurement are in the test phase of a software development life cycle, because it is typically in this phase where the faults are classified into their actual and expected identification phases.

The time between when a fault was inserted and found is commonly referred to as 'fault latency' [138]. Figure 5.1 shows the difference between fault latency and FST [79, 80]. As is clear from this figure, the FST measurement evaluates when it is cost efficient to find a certain fault. To be able to specify this, the organization must first determine what should be tested in which phase [79, 80].

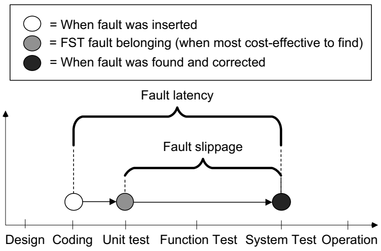

○ = When fault was inserted
● = FST fault belonging (when most cost-effective to find)
● = When fault was found and corrected
Fault latency
Fault slippage
Design
Coding
Unit test
Function Test
System Test
Operation

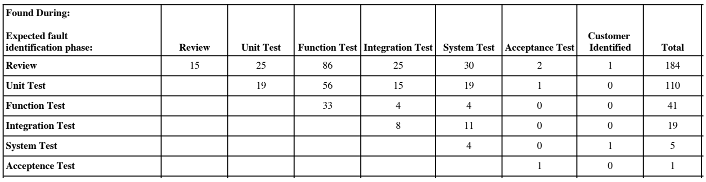
Figure 5.1: Difference between fault latency and FST.

Studies on multiple projects executed within several different organizations at Ericsson [79] showed that FST measurement has some promising advantages:

1. FST can prioritize which phases and activities to improve.

2. The FST measurement approach can assess to which degree a process achieves early and cost-effective software fault detection (one of the studies indicated that it is possible to obtain good indications of the quality of the test process already when 20–30% of the faults have been found).

Figure 5.2 shows a snippet of a faults-slip-through matrix showing the faults slipping through to later phases. The columns in Figure 5.2 represent the phases in which the faults were found (*Found During*) whereas the rows represent the phases where the faults should have been found (*Expected fault identification phase*). For example, 56 of the faults that were found in the function test should have been found during the unit test.



Found During: **Expected fault identification phase:**	**Review**	**Unit Test**	**Function Test**	**Integration Test**	**System Test**	**Acceptance Test**	**Customer Identified**	**Total**
Review	15	25	86	25	30	2	1	184
Unit Test		19	56	15	19	1	0	110
Function Test			33	4	4	0	0	41
Integration Test				8	11	0	0	19
System Test					4	0	1	5
Acceptence Test						1	0	1

Figure 5.2: An example FST matrix.

Apart from the studies done by Damm [79, 80], there are other studies presenting successful industrial implementation of FST measurements. Two such cases are the FST implementations at Ericsson Nikola Tesla [138, 328]. They started collecting FST measurements in all development projects from the middle of the year 2006. The results were encouraging with a decrease of fault-slippage to customers, improvements of test configurations and improvements of test cases used in the verification phase of the projects.

Considering the initial successful results of implementing FST measurement across different organizations within Ericsson, our industrial partner became interested in investigating how to use FST measurement to provide additional decision-support for project management. For example, Staron and Meding [303] highlight that the prediction of the number of faults slipping through can be a refinement to their proposed approach for predicting the number of defects in the defect database. Similarly Damm [79] highlight that FST measurement can potentially be used as a support tool in software fault predictions. This additional decision-support is to make the software development more predictable [275].

The number of faults found by the test team impacts whether or not a project would be completed on schedule and with a certain quality. The project manager has to balance the resources, not only for fixing the identified faults, but also to implement any new functionality. This balance has to be distributed correctly on a weekly or a monthly basis. Any failure to achieve this balance would mean that either the project team is late with the project delivery or the team resources are kept underutilized.

In this chapter, we focus on predicting the number of faults slipping through to different test phases, multiple weeks in advance (a quantitative modeling task). We compare a variety of prediction techniques[2]. Since there is a general lack of empirical evaluation of expert judgement [66, 317] and due to the fact that predictions regarding software quality are based on expert judgements at our organization, and we would argue in industry in general, we specifically compare human expert predictions with these techniques. Thus the motivation of doing this study is to:

- avoid predictable pitfalls like effort/schedule overruns, underutilization of resources and a large percentage of rework.
- provide better decision-support to the project manager so that faults are prevented early in the software development process.
- prioritize which phases and activities to improve.

We also include results from a survey based on an industrial questionnaire. The motivation is to let the industrial experts:

- evaluate the usefulness of predicting FST in different test phases.
- determine the criteria that make prediction techniques usable in industrial practice.

The quantitative data modeling make use of several independent variables at the project level, i.e., variables depicting work status, testing progress status and fault-inflow. The dependent variable of interest is then the number of faults slipping through to various test phases, predicted multiple weeks in advance.

Hence, we are interested in answering the following research questions:

RQ1 Is it and how useful is it to industrial software engineers to predict the number of faults slipping through to different test phases?

RQ2 Can other techniques better predict the number of faults slipping through to different test phases than human expert judgement?

RQ3 How do different techniques compare in FST prediction performance?

[2] statistical techniques (multiple regression, pace regression), tree-structured techniques (M5P, REPTree), nearest neighbor techniques (K-Star, K-nearest neighbor), ensemble techniques (bagging and rotation forest), machine-learning techniques (support vector machines and back-propagation artificial neural networks), search-based techniques (genetic programming, artificial immune recognition systems, particle-swarm optimization based artificial neural networks and gene-expression programming) and expert judgement

RQ4 What criteria are important in making prediction techniques usable in industrial practice?

The mapping between the research questions and the research methodology used is given in Table 5.1.

Table 5.1: Mapping between the research questions and the research methodology.

RQ	Research methodology
RQ1	Qualitative survey based on an industrial questionnaire
RQ2	Quantitative data modeling
RQ3	Quantitative data modeling
RQ4	Qualitative survey based on an industrial questionnaire

The data used in the quantitative data modeling comes from large and complex software projects from the telecommunications industry, as our objective is to come up with results that are representative of real industrial use. Also large-scale projects offer different kinds of challenges, e.g., the factors affecting the projects are diverse and many, data is distributed across different systems and success is dependent on the effort of many resources. Moreover, a large project constitutes a less predictable environment and there is a lack of research on how to use predictive models in such an environment [156].

The rest of the chapter is organized as follows. Section 5.2 summarizes the related work. Section 5.3 describes the study context, variables selection, the test phases under consideration, the performance evaluation measures and the techniques used. Section 5.4 presents a quantitative evaluation of various techniques for the prediction task while the results of the industrial survey are presented in Section 5.5. The results from the quantitative evaluation of different models and the industrial survey are discussed in Section 5.6 while the study validity threats are given in Section 5.7. The chapter is concluded in Section 5.8 while Appendix 10.4 outlines the parameter settings for the different techniques.

5.2 Related work

Due to the definition of software quality in many different ways, previous studies have focused on predicting different but related dependent variables of interest; examples include predicting for defect density [240, 252], software defect content estimation [47, 339], fault-proneness [26, 209] and software reliability prediction in terms of time-to-failure [223]. In addition, several independent variables have been used to predict the above dependent variables of interest; examples include prediction using size and complexity metrics [121], testing metrics [317, 325] and organizational metrics [253]. The actual prediction is performed using a variety of approaches, and can broadly be classified into statistical regression techniques, machine learning approaches and mixed algorithms [67]. Increasingly, evolutionary and bio-inspired approaches are being used for software quality classification [6, 217] while expert judgement is used in very few studies [317, 357].

For a more detailed overview of related work on software fault prediction studies, the reader is referred to [66, 100, 285, 310, 329].

This study is different from the above software quality evaluation studies. First, this study takes a mixed research methodology approach [77], where both quantitative data modeling and qualitative descriptive survey results are presented. Second, the dependent variable of interest for the quantitative data modeling is the number of faults slipping through to various test phases, with the aim of taking corrective actions for avoiding unnecessary rework late in software testing. Third, the independent variables of interest for the quantitative data modeling are diverse and at the *project level*, i.e., variables depicting work status, testing progress status and fault-inflow. A similar set of variables were used in a study by Staron and Meding [303], but they predicted weekly defect inflow and used different techniques. Fourth, for the sake of comparison, we include a variety of carefully selected techniques, representing both commonly used and newer approaches. Fifth, the qualitative descriptive survey assesses the usefulness of predicting FST for the industrial software engineers and the criteria that are important in making prediction techniques usable in industrial practice.

Together this means our study is, we would claim, broader and more industrially relevant than previous studies.

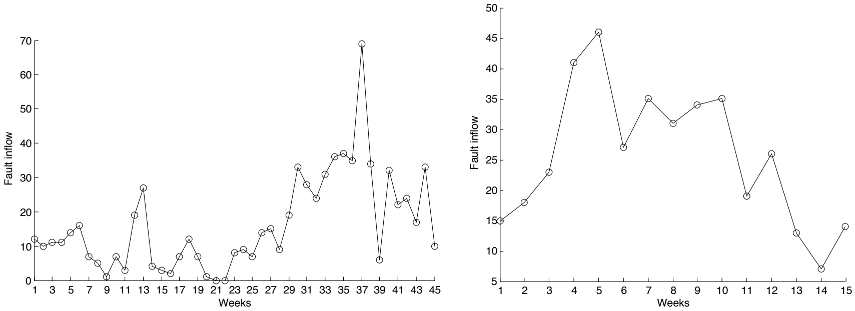
We also would like to mention that this study is an extended version of the authors' earlier conference manuscript [12] where only a limited number of techniques were compared with no evaluation of expert judgement. Furthermore, there was neither an evaluation of the usefulness of FST predictions, nor a discussion of the usability criteria for the prediction techniques.

5.3 Study plan for quantitative data modeling

This section describes the context, independent/dependent variables for the prediction model, the research method, the predictive performance measures and the techniques used for quantitative data modeling. The relevant research questions are RQ2 and RQ3.

5.3.1 Study context

As given in Section 5.1, our context is large and complex software projects in the telecommunications industry. Our subject company develops mobile platforms and wireless semiconductors. The projects are aimed at developing platforms introducing new radio access technologies written using the C programming language. The average number of persons involved in these projects is approximately 250. The data from one of the projects is used as a baseline to train the models while the data from the second project is used to evaluate the models' results. We have data from 45 weeks of the baseline project to train the models while we evaluate the results on data from 15 weeks of an on-going project. Figure 5.3 shows the number of faults occurring per week for the training and the testing set.



(a) Training set.

(b) Testing set.

Figure 5.3: Number of fault occurrences per week for the training set and the testing set.

The management of these projects follow the company's general project model called PROPS (PROfessional Project Steering). PROPS is based on the concepts of tollgates, milestones, steering points and check-points to manage and control project deliverables. Tollgates represent long-term business decisions while milestones are predefined events representing intermediate objectives at the operating work level. The monitoring of these milestones is an important element of the project management model. Steering points are defined to coordinate multiple parallel platform projects, e.g., handling priorities between different platform projects. The checkpoints are defined in the development process to define the work status in a process. Multiple checkpoints might have to be passed for reaching a certain milestone. Figure 5.4 shows an

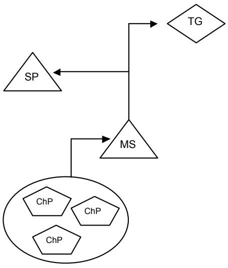


Figure 5.4: PROPS concepts used in the subject company; TG, SP, MS, ChP are short for tollgate, steering point, milestone and checkpoint respectively.

abstract view of these concepts.

At the operative work level, the software development is structured around *work packages*. These works packages are defined during the project planning phase. The work packages are defined to implement change requests or a subset of a use-case, thus the definition of work packages is driven by the functionality to be developed. An essential feature of work packages is that it allows for simultaneous work on different components of the project at the same time by multiple teams.

Since different components might get affected by developing a single work package, therefore it is difficult to obtain consistent metrics at the component level. The structure of a project into work packages present an obvious choice of selecting variables for the prediction models since the metrics at work package level are stable and entails a more intuitive meaning for the employees at the subject company.

Figure 5.5 gives an overview of how a given project is divided into work packages that affects multiple components. The division of an overall system into sub-systems is driven by design and architectural constraints.

5.3.2 Variables selection

At our subject company the work status of various work packages is grouped using a graphical integration plan (GIP) document. The GIP maps the work packages' status over multiple time-lines that might indicate different phases of software testing or overall project progress. There are different *status rankings* of the work packages, e.g., number of work packages planned to be delivered for system integration testing. A

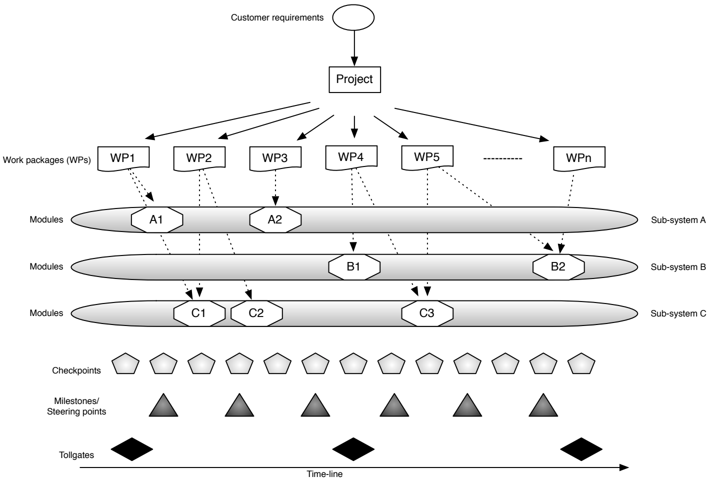


Figure 5.5: Division of requirements into work packages and modules (work model), thereby achieving tollgates, milestones/steering points and checkpoints (management model).

snippet of a GIP is shown in Figure 5.6.

The variables of interest in this study are divided into four sets (Table 5.2), i.e., fault in-flow, status rankings of work packages, faults-slip-through and test case progress.

Table 5.2: Variables of interest for the prediction models.

No.	Description	Abbreviation	Category
1	Fault in-flow	F. in-flow	Fault-inflow
2	No. of work packages planned for system integration	No. WP. PL. SI	Status rankings of WPs
3	No. of work packages delivered to system integration	No. WP. DEL. SI	
4	No. of work packages tested by system integration	No. WP Tested. SI	
5	No. of faults slipping through to all of the test phases	No. FST	FST
6	No. of faults slipping through to the unit test	FST-Unit	
7	No. of faults slipping through to the function test	FST-Func	
8	No. of faults slipping through to the integration test	FST-Integ	
9	No. of faults slipping through to the system test	FST-Sys	
10	No. of system test cases planned	No. System. TCs. PL	TC progress
11	No. of system test cases executed	No. System. TCs. Exec.	
12	No. of interoperability test cases planned	No. IOT TCs. PL	
13	No. of interoperability test cases executed	No. IOT TCs. Exec.	
14	No. of network signaling test cases planned	No. NS TCs. PL	
15	No. of network signaling test cases executed	No. NS TCs. Exec.	

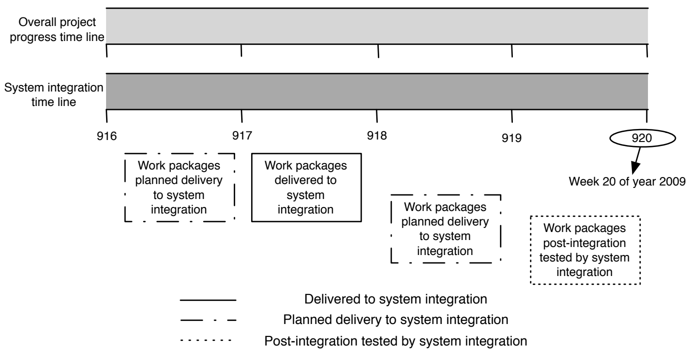

Overall project progress time line
System integration time line
916
917
918
919
920
Week 20 of year 2009
Work packages planned delivery to system integration
Work packages delivered to system integration
Work packages planned delivery to system integration
Work packages post-integration tested by system integration
Delivered to system integration
Planned delivery to system integration
Post-integration tested by system integration

Figure 5.6: The graphical integration plan showing the status of various work packages over multiple time-lines.

During the project life cycle there are certain status rankings related to the work packages (shown under the category of ‘status rankings of WPs’ in Table 5.2) that influence fault-inflow, i.e., the number of faults found in the consecutive project weeks. The information on these status rankings is also conveniently extracted from the GIP, which is a general planning document at the company. Another important set of variables for our prediction models is the actual test case (TC) progress data, shown under the category of ‘TC progress’ in Table 5.2, which have a more direct influence on the fault in-flow. The information on the number of test cases planned and executed at different test phases is readily available from an automated report generation tool that uses data from an internally developed system for fault logging. These variables, along with the status rankings of the work packages, influence the fault-inflow; so we monitor the fault-inflow as another variable for our prediction models. Another set of variables representing the output is the number of faults that slipped-through to the unit, function, integration and system test phases, indicated under the category ‘FST’ in Table 5.2. We also recorded the accumulated number of faults slipping through to all the test phases. All of the above measurements were collected at the subject company on a weekly basis.

5.3.3 Test phases under consideration

Software testing is usually performed at different levels, i.e., at the level of a single component, a group of such components or a complete system [141]. These different levels are termed as *test phases* in our subject company therefore we stick to calling them test phases throughout the chapter. The purpose of different test phases, as defined at our subject company, is given below:

- Unit: To find faults in component internal functional behavior e.g., memory leaks.
- Function: To find faults in functional behavior involving multiple components.
- Integration: To find configuration, merge and portability faults.
- System: To find faults in system functions, performance and concurrency.

Some of these earlier test levels are composed of constituent test activities that jointly make up the higher-order test levels. The following is the division of test levels (i.e., unit, function, integration and system) into constituent activities at our subject company:

- Unit: Hardware development, component test.
- Function: Function test.
- Integration: Integration of components to functions, integration test.
- System: System test, delivery test.

Our focus is then to predict the number of faults slipping through to each of these test phases.

5.3.4 Performance evaluation measures and prediction techniques

The evaluation of predictive performance of various techniques is done using measures of predictive accuracy and goodness of fit.

- The predictive accuracy of different techniques is compared using absolute residuals (i.e., |actual-predicted|) [193, 270, 289].
- The goodness of fit of the results from different techniques is assessed using the two-sample two-sided Kolmogorov-Smirnov (K-S) test at $\alpha = 0.05$. The details regarding this test are given in Section 3.5.1 of this thesis.

We consider a technique better if it performs well for *both* predictive accuracy and goodness of fit. For instance, if no statistically significant differences are found between techniques *A* and *B* for predictive accuracy, but technique *B* has statistically significant goodness of fit in comparison with technique *A*, then technique *B* is declared as better. This is a kind of multi-criteria based evaluation system, a concept similar to the one presented by Lavesson et al. in [206].

A brief description of each of the different techniques used in this study appears in Table 5.3

5.4 Analysis and interpretation

This section describes the quantitative analysis helping us answer RQ2 and RQ3.

5.4.1 Analyzing dependencies among variables

Before applying the specific techniques for prediction, we analyzed the dependencies among variables (see Table 5.2) using scatter plots. We were especially interested in visualizing:

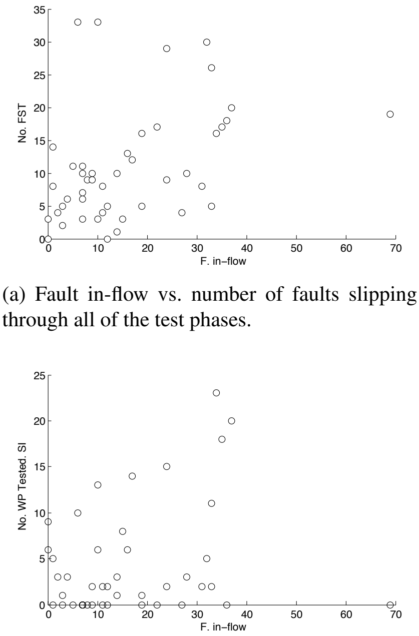
- The relationship between the measures of status rankings of work packages.
- The relationship between the measures of test case progress.
- Fault in-flow vs. the rest of the measures related to status rankings of work packages and test case progress.

The pair-wise scatter plots of the above attributes showed a tendency of non-linear relationship. Two of these scatter plots are shown in Figure 5.7 for *fault in-flow* vs. *number of faults slipping through all of the test phases* (Figure 5.7a) and *fault in-flow* vs. *number of work packages tested by system integration* (Figure 5.7b).

After getting a sense of the relationships among the variables, we used kernel principal component analysis (KPCA) [63] to reduce the number of independent variables to a smaller set that would still capture the original information in terms of explained variance in the data set. The role of original variables in determining the new factors (principal components) is determined by loading factors. Variables with high loadings contribute more in explaining the variance. The results of applying the Gaussian kernel, KPCA (Table 5.4) showed that the first four components explained 97% of the variability in the data set. We did not include the faults-slip-through measures in the

Table 5.3: Prediction techniques used in this study.

Brief overview	Prediction technique
Statistical techniques	
MR is an extension of simple least-square regression for more than one independent (predictor) variables to estimate the values of the dependent (criterion) variable. More information on MR is available in [159]. PR decomposes an initially estimated model (the ordinary least square estimator) into statistically independent dimensional models. These dimensional models are then used to obtain an estimate of the true effects in each individual dimension. More details on PR in [334].	Multiple regression (MR), Pace regression (PR)
Tree-structured techniques	
M5P is tree-induction for regression models. A decision tree induction algorithm is used to build a tree, but instead of maximizing the information gain at each inner node, a splitting criterion is used that minimizes the intra-subset variation in the class values down each branch. More information on M5P is available in [335]. REPTree builds a decision/regression tree using the information gain/variance and prunes it using reduced-error pruning with back-fitting. More information on REPTree is available in Weka documentation at [338].	M5P, REPTree
Nearest neighbor techniques	
Both K* and Knn techniques are analogy-based methods which considers K most similar examples for performing classification. The similarity is measured in K* using an entropy-based distance function (more information in [70] while an euclidean distance is used in knn (more information in [14])	*K*-Star (K*), *K*-nearest neighbor (Knn)
Ensemble techniques	
Ensemble techniques include several base leaners and a voting procedure is used for final classification and an average for a numeric prediction. Bagging generates multiple versions of a predictor and an aggregated average over versions give a numeric outcome. More information in [42]. RF splits the feature set into k-subsets and principal component analysis is applied to each subset. K-axis rotation forms new features for the base-learner. More information in [282].	Bagging, rotation forest (RF)
Machine-learning techniques	
Details regarding SVM algorithm and ANN appear in Sections 4.6.3 and 4.6.2 of this thesis respectively.	Support vector machines (SVM), back-propagation artificial neural networks (ANN)
Search-based techniques	
Search-based techniques model a problem in terms of an evaluation function and then uses a search technique to minimize or maximize that function. Details regarding GP algorithm appear in Section 3.3 of this thesis. In GEP, the individuals making-up the search space (called population) are encoded as linear strings of fixed length that are later expressed as nonlinear expression trees of different sizes and shapes. More information about GEP can be found in [103]. AIRS is inspired by the processes of vertebrate immune system, specifically how B and T lymphocytes improve their response to antigens over time (called affinity maturation). The details of the different steps of the AIRS algorithm can be found in [336]. PSO-ANN uses a particle swarm optimization (PSO) algorithm for training an ANN. PSO is inspired by the coordinated search of food by a swarm of birds. A swarm of particles move through a multidimensional search space for finding global optimum. Trelea [321] proposed several improvements to a basic PSO, called Trelea I and Trelea II depending upon the values of parameters. The PSO version Trelea II is used in this study. More information on PSO-ANN is available at [147].	Genetic programming (GP), gene expression programming (GEP), artificial immune recognition systems (AIRS), particle swarm optimization based artificial neural network (PSO-ANN)
Expert judgement	
An estimate/prediction is based on the experience of one or more people who are familiar with the development of software applications similar to that currently being predicted [139]. The expert in this study has 20 years of work experience and currently holds the designation of systems verification process leader at our subject organization.	Expert judgement



(a) Fault in-flow vs. number of faults slipping through all of the test phases.

(b) Fault in-flow vs. number of work packages tested by system integration.

Figure 5.7: Example scatter plots.

KPCA since these are the attributes we are interested in *predicting*. In each of the four components, all the variables contributed with different loadings, with the exception of two, namely *number of network signaling test cases planned* and *number of network signaling test cases executed*. Hence, we excluded these two variables and use the rest for predicting the faults-slip-through in different test phases.

Table 5.4: The loadings and explained variance from four principal components.

	Variance explained	Variable loadings. The variable names use abbreviations given in Table 5.2										
		F. inflow	No. WP. PL. SI	No. WP. DEL. SI	No. WP Tested. SI	No. FST	No. System. TCs. PL	No. System. TCs. Exec.	No. IOT TCs. PL	No. IOT TCs. Exec.	No. NS TCs. PL	No. NS TCs. Exec.
Component 1	51.61%	0.60	0.02	0.01	0.09	0.38	0.61	0.34	0.02	0.02	0	0
Component 2	31.07%	0.75	-0.02	0.01	0.13	-0.01	-0.57	-0.32	0.03	0.03	0	0
Component 3	9.88%	-0.29	0.01	0	0.52	0.75	-0.20	-0.11	0.11	0.12	0	0
Component 4	4.64%	0	0	0.02	0.83	-0.50	0.19	0	-0.07	-0.07	0	0

Specifically, for predicting the faults-slippage to unit test, we use the fault in-flow, work-package status rankings and test case progress metrics. For predicting the faults-slippage to subsequent test phases we also include the faults-slippage for the proceeding test phase; for instance when predicting the faults-slip-through at the function test phase, we also use the faults-slip-through at unit test phase as an independent variable along with fault inflow, work-package status rankings and test case progress metrics.

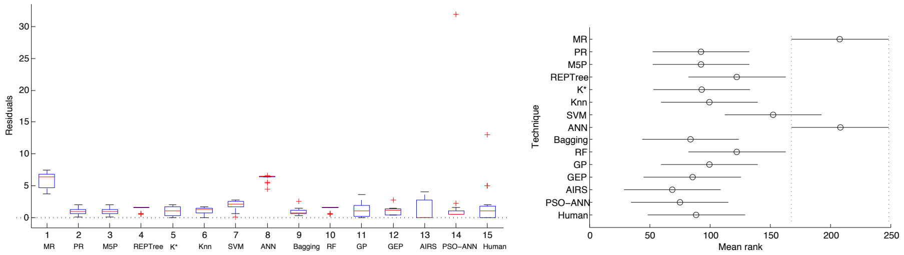
5.4.2 Performance evaluation of techniques for FST prediction

Next we present the results of the performance of different techniques in predicting FST for each test phase that would help us to answer RQ3. As given in Section 5.3.4, we evaluate the prediction performance using the measures for predictive accuracy and goodness of fit.

The common analysis procedure to follow is to compare the box-plots of the absolute residuals for different prediction techniques. But since box-plots cannot confirm whether one prediction technique is significantly better than another, we use a statistical test (parametric or a non-parametric test—depending upon whether the assumptions of the test are satisfied) for testing the equality of population medians among groups of prediction techniques. Upon the rejection of the null hypothesis of equal population medians, a multiple comparisons (post-hoc) test is performed on the group medians to determine which means differ. Finally, we proceed with assessing the goodness of fit using the K-S test described in Section 5.3.4.

Prediction of FST at the unit test phase

The box-plots of absolute residuals for predicting FST at the unit test phase for different techniques is shown in Figure 5.8a. The box-plot having the median value close to the 0 mark on the *y*-axis (shown as a dotted horizontal line in Figure 5.8a) and a smaller spread of the distribution indicate better predictive accuracy. Keeping in view these two properties of the box-plots, there seems to be only a marginal difference in the residual box-plots of PR, M5P, Knn, Bagging, GEP and PSO-ANN. AIRS has a median at the 0 mark but shows larger spread in comparison with other techniques. The human/expert prediction also shows a larger spread but smaller than AIRS. Two outliers for the human prediction are extreme as compared to the one extreme outlier for PSO-ANN. Predictions from MR, SVM and ANN appear to be farther away from the 0 mark on the *y*-axis, an indication that the predictions are not closely matching the actual FST values.



(a) Box-plots of the residuals for each technique in predicting FST at the unit test phase.

(b) Results of the multiple comparisons test with $\alpha = 0.05$ (The vertical dotted lines indicating that 12 techniques have mean ranks significantly different from MR).

Figure 5.8: Results showing box plots of absolute residuals and multiple comparisons of the absolute residuals between all techniques at the unit test phase.

To test for any statistically significant differences in the models' residuals, the non-parametric Kruskal-Wallis test was used to examine any statistical differences between the residuals and to confirm the trend observed from the box-plots. The skewness in the residual box-plots for some techniques motivated the use of the non-parametric test. The result of the Kruskal-Wallis test ($p = 3.2e^{-14}$) suggested that it is possible to reject the null hypothesis of all samples being drawn from the same population at significance level, $\alpha = 0.05$. This is to suggest that at least one sample median is significantly different from the others. In order to determine which pairs are significantly different, we apply a multiple comparisons test (Tuckey-Kramer, $\alpha = 0.05$). The results of the multiple comparisons are displayed using a graph given in Figure 5.8b. The mean of each prediction technique is represented by a circle while the straight lines on both sides of the circle represents an interval. The means of two prediction techniques are significantly different if their intervals are disjoint and are not significantly different if their intervals overlap. For illustrative purposes, Figure 5.8b shows vertical dotted lines for MR. There are two other techniques (SVM and ANN) where either of these two dotted lines cut through their intervals, showing that the means for MR, SVM and ANN are not significantly different. It is interesting to observe that there is only a single technique (AIRS) whose mean is significantly different (and better) than all these three techniques (i.e., MR, SVM and ANN). There are, however, no significant pair-wise differences between the means of AIRS and rest of the techniques (i.e., PR, M5P, REPTree, K*, Knn, Bagging, RF, GP, GEP, PSO-ANN, Human). Human predictions, on the other hand, are significantly different and better than two of the least accurate techniques (MR, ANN).

The K-S test result for measuring the goodness of fit for predictions from each technique relative to the actual FST at the unit test phase appear in Table 5.5. The techniques having statistically significant goodness of fit are shown in bold (AIRS and Human). Figure 5.9 shows the plot of AIRS, human and actual FST at the unit test phase. The statistically significant goodness of fit for AIRS and human can be attributed to the exact match of actual FST data on 9 out of 15 instances for AIRS and 5 out of 15 instances for the human. However, the human prediction is off by large values in the last three weeks that can also be seen as extreme outliers in Figure 5.8a.

Table 5.5: Two-sample two sided K-S test results for predicting FST at the unit test phase with critical value $J_{0.05} = 0.5$.

K-S test statistic, *J*														
MR	PR	M5P	REPTree	K*	Knn	SVM	ANN	Bagging	RF	GP	GEP	AIRS	PSO-ANN	Human
1	0.60	0.60	0.93	0.53	0.80	0.87	1	0.80	0.93	0.53	0.80	**0.27**	0.73	**0.33**

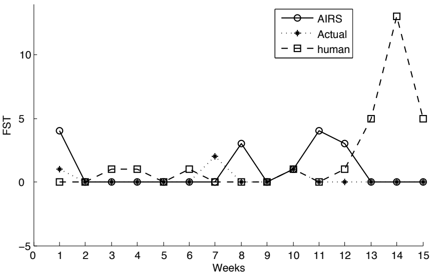
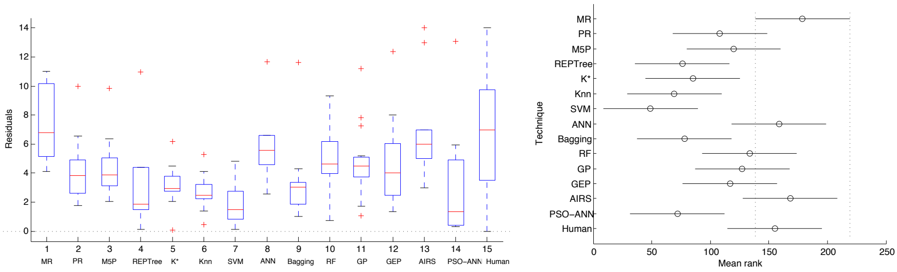


Figure 5.9: Plot of the predicted vs. the actual FST values at the unit test phase for techniques having significant goodness of fit.

In summary, in terms of predictive accuracy, AIRS showed significantly different absolute residuals in comparison with the three least performing techniques for predicting FST at the unit test phase. But then there were found no significant differences between the absolute residuals of AIRS and rest of the 11 techniques. Human predictions showed significantly different absolute residuals in comparison with the two least performing techniques for predicting FST at the unit test phase. For goodness of fit, AIRS and human predictions were found to be statistically significant, though the human predictions resulted in extreme values later in the prediction period.

Prediction of FST at the function test phase

The box-plots of absolute residuals for predicting FST at the function test phase for different techniques is shown in Figure 5.10a. We can observe that there is a greater spread of distribution for each of the techniques as compared with those at the unit test phase. The box-plots for each of the techniques are also farther away from the 0 mark on the y-axis, with PSO-ANN and SVM having the median closet of all to the 0 mark on the y-axis. Human and MR prediction shows the greatest spread of distributions while the box-plots of PR, M5P, K*, Knn and Bagging show only a marginal difference. The result of the Kruskal-Wallis test ($p = 1.6e^{-11}$) at $\alpha = 0.05$ suggested that at least one sample median is significantly different from the others. Subsequently, the results of the multiple comparisons test (Tuckey-Kramer, $\alpha = 0.05$) appear in Figure 5.10b. The absolute residuals of MR and human are not significantly different (as their intervals overlap), a confirmation of the trend observed from the box-plots. Two of the better techniques having lower medians are SVM and PSO-ANN. There are no significant differences between the two. Also there are no significant pair-wise differences between SVM and each one of: PR, M5P, REPTree, K*, Knn, Bagging, GP, GEP.



(a) Box-plots of the residuals for each technique in predicting FST at the function test phase.

(b) Results of the multiple comparisons test with $\alpha = 0.05$ (The vertical dotted lines indicating that 6 techniques have mean ranks significantly different from MR).

Figure 5.10: Results showing box plots of absolute residuals and multiple comparisons of the absolute residuals between all techniques at the function test phase.

The K-S test result for measuring the goodness of fit for predictions from each technique relative to the actual FST at the function test phase appear in Table 5.6. SVM and PSO-ANN show statistically significant goodness of fit. Figure 5.11 shows the line plots of SVM and PSO-ANN with the actual FST at the function test phase.

Table 5.6: Two-sample two sided K-S test results for predicting FST at the function test phase with critical value $J_{0.05} = 0.5$.

K-S test statistic, *J*														
MR	PR	M5P	REPTree	K*	Knn	SVM	ANN	Bagging	RF	GP	GEP	AIRS	PSO-ANN	Human
1	0.93	0.93	0.73	0.93	0.93	**0.4**	1	0.80	0.93	0.93	0.93	0.93	**0.4**	0.73

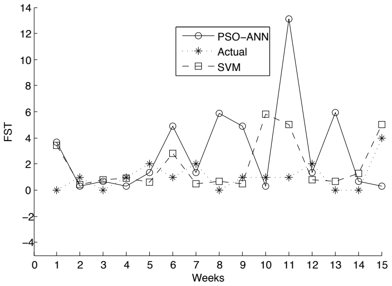


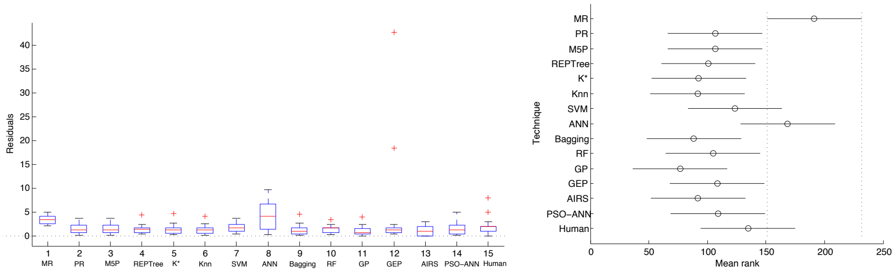
Figure 5.11: Plot of the predicted vs. the actual FST values at the function test phase for techniques having significant goodness of fit.

SVM appears to behave in Figure 5.11 since there are no high peaks showing outliers (as is the case with PSO-ANN in Week 11).

In summary, in terms of predictive accuracy, residual box-plots indicate that SVM and PSO-ANN are better at predicting FST at the function test phase but there are no significant differences found with the majority of the other techniques. Also MR and human predictions are significantly worse than seemingly better SVM and PSO-ANN. SVM and PSO-ANN also show statistically significant goodness of fit in comparison with other techniques.

Prediction of FST at the integration test phase

The box-plots of absolute residuals for predicting FST at the integration test phase for different techniques is shown in Figure 5.12a. We can observe that there is a smaller spread of distribution for each technique as compared with the box-plots for function test. An exception is ANN whose box-plot is more spread out than other techniques. In terms of the median being close to the 0 mark on the *y*-axis, Bagging and GP appear to be promising, though there seem to be only marginal differences in comparison with



(a) Box-plots of the residuals for each technique in predicting FST at the integration test phase.

(b) Results of the multiple comparisons test with $\alpha = 0.05$ (The vertical dotted lines indicating that eleven techniques have mean ranks significantly different from MR).

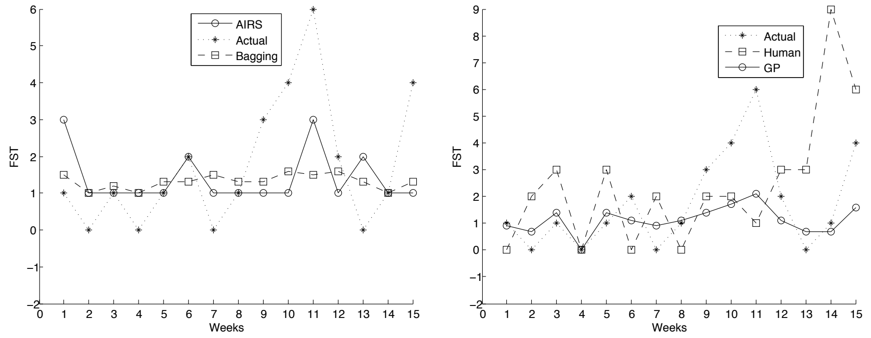
Figure 5.12: Results showing box plots of absolute residuals and multiple comparisons of the absolute residuals between all techniques at the integration test phase.

PR, M5P, REPTree and PSO-ANN. GEP and human each show two extreme outliers. The result of the Kruskal-Wallis test ($p = 1.7e^{-5}$) at $\alpha = 0.05$ suggested that at least one sample median is significantly different from the others. Subsequently, the results of the multiple comparisons test (Tuckey-Kramer, $\alpha = 0.05$) appear in Figure 5.12b. The mean rank for MR is not significantly different than the ones for SVM, ANN and the human. GP has the mean rank that is significantly different than MR and ANN, the two least performing techniques. However, there are not any pair-wise significant differences between the absolute residuals for GP and each one of: PR, M5P, REPTree, K*, Knn, SVM, Bagging, RF, GEP, AIRS, PSO-ANN and Human.

The K-S test result for measuring the goodness of fit for predictions from each technique relative to the actual FST at the integration test phase appear in Table 5.7. Bagging, GP, AIRS and human predictions show statistically significant goodness of fit. Figure 5.13 show the line plots of Bagging, GP, AIRS and the human predictions.

Table 5.7: Two-sample two sided K-S test results for predicting FST at the integration test phase with critical value $J_{0.05}$ = 0.5.

K-S test statistic, J														
MR	PR	M5P	REPTree	K*	Knn	SVM	ANN	Bagging	RF	GP	GEP	AIRS	PSO-ANN	Human
0.73	0.60	0.60	0.60	0.60	0.60	0.60	0.60	**0.40**	0.60	**0.33**	0.60	**0.27**	0.60	**0.27**



(a) Plot of the actual vs. predicted FST values (AIRS and Bagging) at the integration test phase.

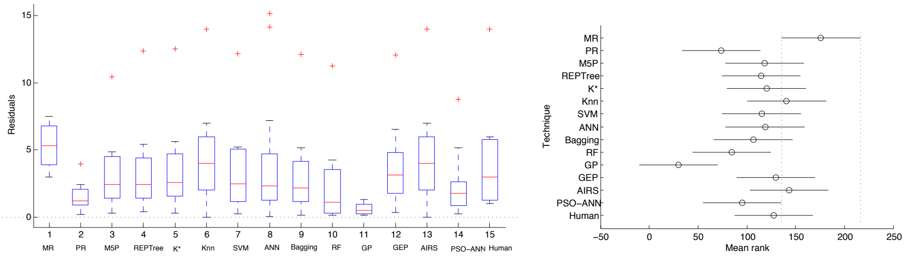
(b) Plot of the actual vs. predicted FST values (Human and GP) at the integration test phase.

Figure 5.13: Plot of the predicted vs. the actual FST values at the integration test phase for techniques having significant goodness of fit.

In summary, in terms of predictive accuracy, MR and ANN appear to be the two least performing techniques for predicting FST at the integration test phase, while there were no statistically significant differences between the majority of the techniques. Bagging, GP, AIRS and human predictions show statistically significant goodness of fit in comparison with other techniques.

Prediction of FST at the system test phase

The box-plots of absolute residuals for predicting FST at the system test phase for different techniques is shown in Figure 5.14a. We can observe that there are certain techniques that appear to do better. These are PR, RF and GP. The box-plots of these three techniques have medians closer to the 0 mark on the y-axis, with GP being the closest. GP also show the smallest distribution as compared with PR and RF. For the rest of the techniques, there is a greater variance in their box-plots with outliers. MR, Knn, AIRS and human box-plots seem to be worse, both in terms of the position of the median and the spread of the distribution. The result of the Kruskal-Wallis test ($p = 5.6e^{-7}$) at $\alpha = 0.05$ suggested that at least one sample median is significantly different from the others. Subsequently, the results of the multiple comparisons test (Tuckey-Kramer, $\alpha = 0.05$) appear in Figure 5.14b. The technique with smallest mean rank is GP and there are no pair-wise significant differences between GP and any of the techniques: PR, Bagging, RF and PSO-ANN. This finding also confirms the trend from the box-plots. MR is the worst performing technique and there are no pair-wise



(a) Box-plots of the residuals for each technique in predicting FST at the system test phase.

(b) Results of the multiple comparisons test with $\alpha = 0.05$ (The vertical dotted lines indicating that 3 techniques have mean ranks significantly different from MR).

Figure 5.14: Results showing box plots of absolute residuals and multiple comparisons of the absolute residuals between all techniques at the system test phase.

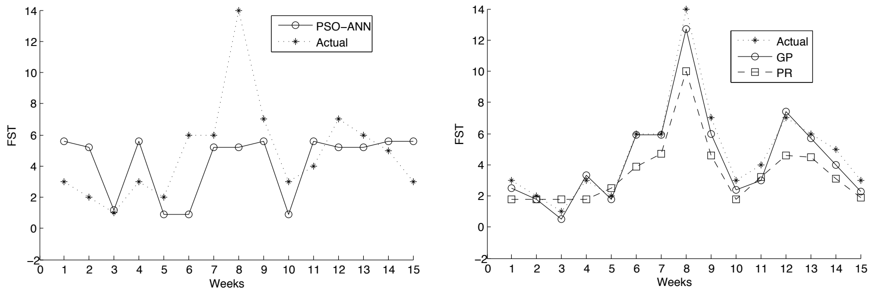
significant differences between MR and any of the techniques: M5P, REPTree, K*, Knn, SVM, ANN, Bagging, GEP, AIRS, PSO-ANN and Human.

The K-S test result for measuring the goodness of fit for predictions from each technique relative to the actual FST at the system test phase appear in Table 5.8. PR, GP and PSO-ANN show statistically significant goodness of fit. Figure 5.15 shows the line plots of PR, GP, PSO-ANN with the actual FST at the system test phase.

Table 5.8: Two-sample two sided K-S test results for predicting FST at the system test phase with critical value $J_{0.05}$ = 0.5.

K-S test statistic, *J*														
MR	PR	M5P	REPTree	K*	Knn	SVM	ANN	Bagging	RF	GP	GEP	AIRS	PSO-ANN	Human
0.67	**0.40**	0.73	0.93	0.93	0.80	0.87	0.47	0.93	0.67	**0.20**	0.80	0.73	**0.40**	0.60

In summary, in terms of predictive accuracy, GP, PR, Bagging, RF and PSO-ANN perform better than the other techniques for predicting FST at the system test phase. PR, GP and PSO-ANN show statistically significant goodness of fit in comparison with other techniques.



(a) Plot of the actual vs. predicted FST values (PSO-ANN) at the system test phase.

(b) Plot of the actual vs. predicted FST values (GP and PR) at the system test phase.

Figure 5.15: Plot of the predicted vs. the actual FST values at the system test phase for techniques having significant goodness of fit.

5.4.3 Performance evaluation of human expert judgement vs. other techniques for FST prediction

The analysis done in the previous Section 5.4.2 would also allow us to answer the RQ2 that questions if other techniques better predict FST than human expert judgement. We now analyze the performance of human expert judgement vs. other techniques for FST prediction at each of the four test phases.

Prediction of FST at the unit test phase

Figure 5.8b shows the results of the multiple comparisons test (Tuckey-Kramer, $\alpha = 0.05$) for FST prediction at the unit test phase. Two techniques have their means significantly different (and worse) than the human expert judgement. These techniques are MR and ANN. Otherwise, there are no significant pair-wise differences between the means of human expert judgement and rest of the techniques.

In terms of goodness of fit, Table 5.5 shows that AIRS and human expert judgement have statistically significant goodness of fit in comparison with other techniques.

Prediction of FST at the function test phase

Figure 5.10b shows the results of the multiple comparisons test (Tuckey-Kramer, $\alpha = 0.05$) for FST prediction at the function test phase. Three techniques have their means significantly different (and better) than the human expert judgement. These techniques are PSO-ANN, SVM and Knn. Otherwise, there are no significant pair-wise differences between the means of human expert judgement and the rest of the techniques.

In terms of goodness of fit, Table 5.6 shows that human expert judgement have no significant goodness of fit in comparison with SVM and PSO-ANN.

Prediction of FST at the integration test phase

Figure 5.12b shows the results of the multiple comparisons test (Tuckey-Kramer, $\alpha = 0.05$) for FST prediction at the integration test phase. No technique has its mean significantly different than the human expert judgement.

In terms of goodness of fit, Table 5.7 shows that Bagging, GP, AIRS and human expert judgement have significant goodness of fit in comparison with rest of the techniques.

Prediction of FST at the system test phase

Figure 5.14b shows the results of the multiple comparisons test (Tuckey-Kramer, $\alpha = 0.05$) for FST prediction at the system test phase. GP has its mean significantly different (and better) than the human expert judgement.

In terms of goodness of fit, Table 5.8 shows that GP and PSO-ANN have significant goodness of fit in comparison with rest of the techniques.

Table 5.9 sums up which techniques are or are not better than human expert judgement in predicting FST at unit, function, integration and system test phases. The dark grey cells in the Table 5.9 refers to techniques that are equally good in predicting FST with the human expert judgement. The light grey cells indicate that the techniques are inferior with respect to the human judgement and the dark grey cells. The white cells indicate that these techniques are better than human expert judgement in predicting FST.

Table 5.9: A summary of techniques that are or are not better than predicting FST at unit, function, integration and system test phases.

Human expert judgement vs.														
	MR	PR	M5P	REPTree	K*	Knn	SVM	ANN	Bagging	RF	GP	GEP	AIRS	PSO-ANN
Unit	░	░	░	░	░	░	░	░	░	░	░	░	▓	░
Function														
Integration	░	░	░	░	░	░	░	░	▓	░	▓	░	▓	░
System														

5.5 Results from an industrial survey

In order to answer RQ1 and RQ4, we conducted an industrial survey based on a questionnaire. The goal of the survey was to evaluate the usefulness and usability of having FST predictions in different test phases. We conducted this survey at companies known to be interested in collecting and reporting FST measurements. The survey involved two industrial organizations and targeted experienced professionals. As such a total of five responses were collected from professionals serving a variety of managerial positions: technical manager test (1), line manager (2), project manager verification (1) and quality coordinator (1).

5.5.1 Survey study design

The survey was conducted based on a questionnaire. It can be considered as a descriptive survey and gives distribution of certain characteristics or attributes, as opposed to explaining or showing causal relationships between variables [263]. Descriptive surveys focus on describing the frequencies of certain events happening regardless of why the observed distribution exists [342].

We did a purposive sampling [254] whereby the respondents were selected based on their interest in reporting the FST metric. A total of five responses were collected from two telecommunications companies. The questionnaire included a precise description of its intent and included explanatory text for completing different parts. The questionnaire was emailed to the contact persons in the two companies and a response was awaited. Mail data collection [21] was therefore the principal data collection mechanism used.

We requested the background information of the respondents in the questionnaire. The respondents' overall work experience varied in the range of 5 to 30 years while the experience in using/reporting FST varied in the range of 2 to 10 years. The frequency of using/reporting FST data also showed considerable variance whereby one respondent reported/used FST on bi-monthly basis while others reported/used them once or twice a year. The respondents' number of years of work experience, the number of years in using/reporting FST and the frequency of using/reporting FST is shown in Figure 5.16.

5.5.2 Assessment of the usefulness of predicting FST

The questionnaire asked multiple questions to evaluate how useful it is to predict FST in different test phases multiple weeks in advance. The Likert [211] format questions were formulated with the following format:

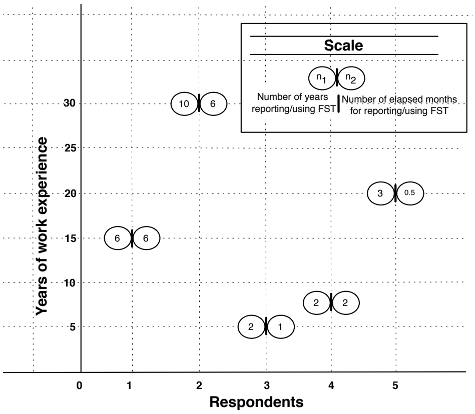


Figure 5.16: Background information regarding the questionnaire respondents.

- Strongly disagree.
- Disagree.
- Neither agree nor disagree.
- Agree.
- Strongly agree.

Additionally, the respondents were given an option of ‘no opinion’. The questions regarding the usefulness of predicting FST were divided into several themes, i.e., usefulness in terms of:

1. Committing realistic deadlines.
2. Realistically planning testing resources for a test assignment.
3. Visualizing ahead in time the expected fault-detection efficiency of test phases.
4. Taking corrective actions in time for the test phase with expected high FST.
5. Reducing the overall cost of testing by improving test effort on phases with high fault slippage.

6. Continuously monitoring and improving the test phase efficiency over time.
7. Reducing the chances of slipping high severity faults to the end customer.
8. Baselining the expected effort to be spent on different test phases.

The questions regarding the above themes were posed in an affirmative sense, i.e., questioning the positive outcome. For example, the question regarding commitment of realistic deadlines was posed as: How much do you agree that accurate prediction of FST *would help* the test team to commit realistic deadlines?, instead of asking: How much do you agree that accurate prediction of FST *would not help* the test team to commit realistic deadlines?

Figure 5.17 illustrates the reported usefulness of predicting FST against the above themes. Overall, the agreement level is high, with most answers in 'agree' and 'strongly agree' levels and none in the 'strongly disagree' level (Figure 5.18). The themes having the most 'strongly agree' votes are:

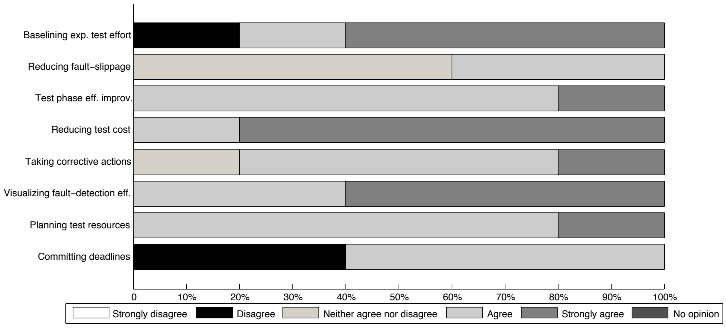


Figure 5.17: The respondents' view of the usefulness of predicting FST in different test phases (questions asked in affirmation).

- Baselining the expected effort to be spent on different test phases.
- Reducing the overall cost of testing by improving test effort on phases with high fault slippage.
- Visualizing ahead in time the expected fault-detection efficiency of test phases.

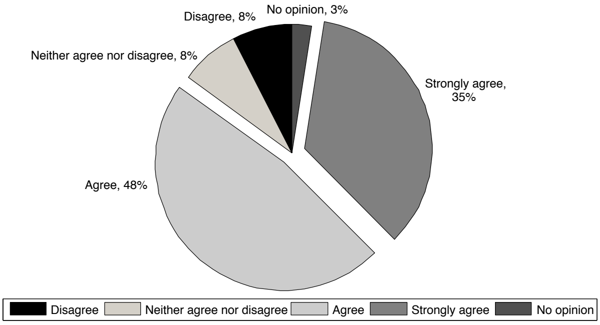


Figure 5.18: Percentage of votes belonging to various levels in evaluating the usefulness of predictions (questions asked in affirmation).

The themes having the 'agree' votes as dominant are:

- Committing realistic deadlines.
- Realistically planning testing resources for a test assignment.
- Taking corrective actions in time for the test phase with expected high FST.
- Continuously monitoring and improving the test phase efficiency over time.

One theme where the majority of the votes are neutral, i.e., 'neither agree nor disagree' is usefulness in terms of reducing high severity fault-slippage to the end customer.

The second set of questions evaluating the usefulness of predicting FST were posed in a negative sense, i.e., questioning the possibility of a negative outcome. The questions were divided into several themes, i.e., questioning that prediction of FST would be counter-productive because:

1. It would be too time-consuming.
2. It would result in major change in existing development process.
3. It would incur huge training cost.
4. It would give us a false sense of security.

The results of questions regarding the above themes are illustrated in Figure 5.19.

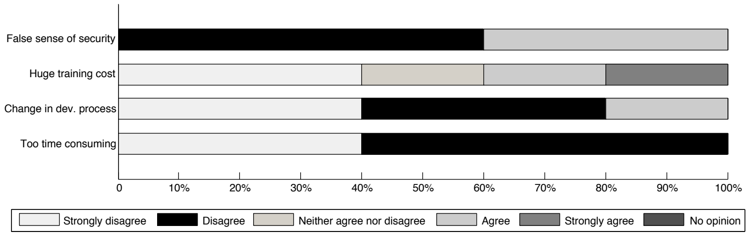


Figure 5.19: The respondents' view of the usefulness of predicting FST in different test phases (questions asked in negation).

Overall, a total of 70% of votes are in disagreement (Figure 5.20).

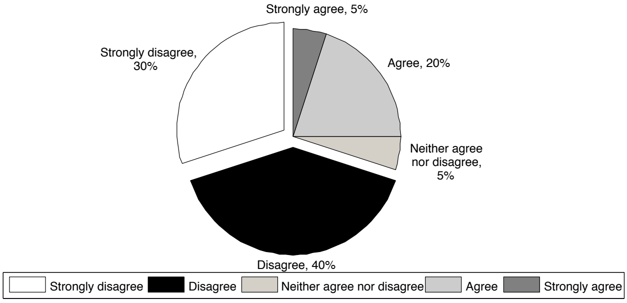


Figure 5.20: Percentage of votes belonging to various levels in evaluating the usefulness of predictions (questions asked in negation).

30% of the votes are in 'neither agree nor disagree' while 40% are in the 'disagree' level. The theme 'Prediction of FST would be counter-productive because it would incur huge training cost' has one vote with level 'strongly agree' and one with level 'agree'. The following themes are dominated by disagreement levels ('strongly disagree' and 'disagree':

- Prediction of FST would be counter-productive because:
  - It would be too time-consuming.
  - It would result in major change in existing development process.

- It would give us false sense of security.

5.5.3 Assessment of the usability of predicting FST

The questionnaire asked multiple questions to evaluate the importance of certain criteria when using a particular technique for predicting FST in different test phases. As with evaluating the usefulness, Likert [211] format questions were formulated, complemented by a 'no opinion' option. The following criteria were used to evaluate the usability of the prediction techniques:

1. The techniques used for prediction need to be part of an automated tool.

2. We need to know the working principles of the prediction technique to understand how is it working.

3. The techniques used for prediction should be able to determine the form of relationship between inputs and outputs rather than being dependent on the user for providing the form of the relationship.

4. Most of the techniques used for prediction requires settings of some parameter values, which are regarded as a hindrance in the use of these techniques.

5. Different techniques apply different mechanisms for predicting an output and the reasoning process might be transparent or black-box. Do you think that the final relationships between inputs and output should be transparent?

The results are illustrated in Figure 5.21. The respondents agreed to the larger extent (80%) that the techniques used for prediction need to be part of an automated tool (40% – 'agree' and 40% – 'strongly agree'). A similar trend was observed with respect to the users being able to know the working principles of the techniques and the prediction technique being able to determine the form of the relationship between the inputs and the outputs (80% of the respondents were in agreement on these two usability aspects). In relation to the parameter settings of various techniques being a hindrance in the use of these techniques, the agreement level is still higher (60%) but for some respondents, it was difficult to answer with certainty (20% – 'no opinion' and 20% – 'neither agree/disagree'). On the question if it is useful to have the final relationships between inputs and outputs being transparent, 80% respondents were in agreement (60% – 'agree' and 20% – 'strongly agree').

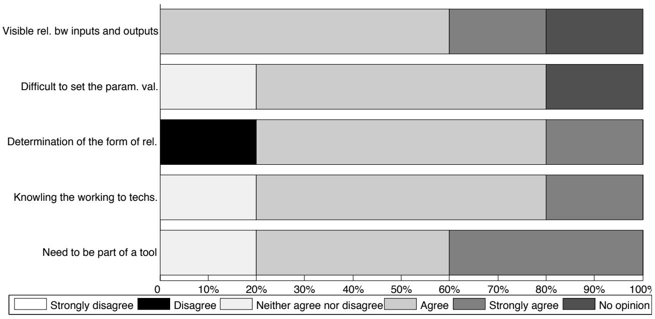


Figure 5.21: The respondents' view of the usability of predicting FST in different test phases.

5.6 Discussion

One of the basic objectives of doing measurements is monitoring of activities so that action can be taken as early as possible to control the final outcome. With this objective in focus, FST metrics work towards the goal of minimization of avoidable rework by finding faults where they are most cost-effective to find. Early prediction of FST at different test phases is an important decision-support to the development team whereby advance notification of improvement potential can be made.

In this chapter we investigated four research questions outlined in Section 5.1. RQ3 investigated the use of a variety of techniques for predicting FST in unit, function, integration and system test phases. The results are evaluated for predictive accuracy (through absolute residuals) and goodness of fit (through the Kolmogorov-Smirnov test). A number of techniques are found to be useful in predicting FST for different test phases (both in terms of predictive accuracy and goodness of fit). RQ2 is concerning a more specific research question that compared human expert judgement with other techniques. The results of this comparison indicate that expert human judgement is better than majority of the techniques at unit and integration test but are far off at function and system test. Hence, human predictions regarding FST lack some consistency. There are indications that a smaller group of techniques might be consistently better in predicting at all the test phases. Following is the list of techniques performing better at various test phases for predicting FST in our study:

1. Unit test – AIRS and human.

2. Function test – SVM and PSO-ANN.

3. Integration test – Bagging, GP, AIRS, human.
4. System test – PR, GP, PSO-ANN.

A trend that can be observed from this list of comparatively better techniques is that there is a representation of search-based techniques in predicting FST at each test phase.

- AIRS is consistently better at – Unit and integration test.
- PSO-ANN is consistently better at – function and system test
- GP is consistently better at – integration and system test.

At this point in time we are not able to provide definitive reasons for why the search-based techniques are able to perform better consistently as compared with other techniques. What we can discuss here are relative merits of the search-based techniques, one or more of which might be responsible for outperforming the other group of techniques.

- Better ability to cope with ill-defined, partial and messy input data [129].
- GP is particularly good at providing small programs that are nearly correct and predictive models are not exceptionally long [129].
- Being non-parametric approaches, the structure of the end solution is not preconceived.
- Entirely data driven approaches and do not include any assumptions about the distribution of the data in its formulation.

GP, in particular, also fulfills two of the usability aspects that showed higher agreement levels among industrial experts, i.e.,

- The techniques used for prediction should be able to determine the form of relationship between inputs and outputs rather than that the technique is dependent on the user providing the form of the relationship.
- The final relationship between the inputs and the outputs should be visible.

The results also argue that there is value in the use of other techniques like human predictions, SVM, Bagging and PR, it is just that these are not as consistent as the search-based techniques.

Another interesting outcome of this study is the performance of search-based techniques (and other better performing techniques) outside their respective training ranges, i.e., the predictions are evaluated for 15 weeks of an on-going project after being trained on another baseline project data. This is to say that the over-fitting is within acceptable limits, and this is particularly encouraging considering the fact that we are dealing with large projects where the degree of variability in fault occurrences can be large. This issue is also related to the amount of data available for training the different techniques, which, in case of large projects is typically available.

Another important aspect of the results is that human predictions were among the better techniques for predicting FST at unit and integration test. In our view, this is also an important outcome and shows that expert opinions perhaps need more consideration that is largely been ignored in empirical studies of software fault predictions [66, 317]. We, therefore, agree with the conclusion of Hughes [139] that expert judgement should be supported by the use of other techniques rather than displacing it. Search-based techniques seem to be an ideal decision-support tool for two reasons:

1. They have performed consistently better than other techniques (Section 5.4.2).
2. Search-based techniques, as part of the more general field of search-based software engineering (SBSE) [128, 131], is inherently concerned with *improving* not with *proving* [129].

As such it is likely that *human-guided* semi-automated search might help get a reasonable solution that incorporates human judgement in the search process. This human-guided search is commonly referred to as 'human-in-the-loop' or 'interactive evolution' [129] and is a promising area of future research. The incorporation of human feedback in the automated search can possibly account for some of the extreme fluctuations in the solely human predictions that are observed for predicting FST at unit and integration test.

We also believe that the selection of predictor variables that are easy to gather (e.g., the project level metrics at the subject company in this study) and that do not conflict with the development life cycle have better chances of industry acceptance. There is evidence to support that general process level metrics are more accurate than code/ structural metrics [26]. A recent study by Afzal [4] has shown that the use of number of faults-slip-through to/from various test phases are able to provide good results for finding fault-prone components at integration and system test phases. However this subject requires further research.

We have also come to realize that the calculation of simple residuals and goodness of fit tests along with statistical testing procedures are a sound way to secure empirical findings where the outcome of interest is numeric rather than binary. An assessment of the qualitative features can then be undertaken as an industrial survey to complement the initial empirical findings.

While working on-site at the subject organization for this research, we realized several organizational factors that influence the success of such a decision-support. Managerial support and an established organizational culture of quantitative decision-making allowed us to gain easy access to data repositories and relevant documentation. Moreover, collection of faults-slip-through data and association of that data to components, was made possible using automated tool support that greatly reduced the time for data collection and ensured data integrity.

5.7 Empirical validity evaluation

We adopted a case study approach in evaluating various techniques for predicting FST in four test phases, while conducting an industrial survey for an assessment of the usefulness of such predictions and the usability of prediction techniques. A controlled experiment was deemed not practical since too many human factors potentially affect fault occurrences.

What follows next is our presentation of the various threats to validity of our study: *Construct validity:* Our choice of selecting project level metrics (Table 5.2) instead of structural code metrics was influenced by multiple factors. First, metrics relevant to work packages (Section 5.3.1) have an intuitive appeal for the employees at the subject company where they can relate FST to the proportion of effort invested. Secondly, the existence of a module in multiple work packages made it difficult to obtain consistent metrics at the component level. Thirdly, the intent of this study is to use project level metrics that are readily available and hence reduces the cost of doing such predictions. In addition, the case study is performed in the same development organization having the identical application domain, so the two projects in focus are characterized by the same set of metrics. *Internal validity:* A potential threat to the internal validity is that the FST data did not consider the severity level of faults, rather treated all faults equally. As for the prediction techniques, the best we could do was to experiment with a variety of parameter values. But we acknowledge that the obtained results could be improved by better optimizing the parameters. *External validity:* The quantitative data modeling was performed on data from a specific company while the questionnaire was filled out by experts from two companies. We have tried to present the context and the processes to the extent possible for fellow researchers to generalize our results. We are also encouraged by the fact that the companies are enterprise-size and have development centers worldwide that follow similar practices. It is therefore likely that the results of this study are useful for them too. A threat to the external validity is that we cannot publicize our industrial data sets due to proprietary concerns. However, the transformed representation of the data can be made available if requested. *Conclusion validity:* We were conscious in using the right statistical test, basing our selection on whether the assumptions of the test were met or not. We used a significance level of 0.05, which is a commonly used significance level for hypothesis testing [158]; however, facing some criticism lately [142].

5.8 Conclusion

In this chapter, we have presented an extensive empirical evaluation of various techniques for predicting the number of faults slipping through to the four test phases of unit, function, integration and system. We also reported on the results of a survey based on an industrial questionnaire that evaluates the usefulness of such a prediction task and evaluates criteria important in making prediction techniques usable in industrial practice.

We find that a number of techniques are found to be useful in such a prediction task, both in terms of predictive accuracy and goodness of fit. However, the group of search-based techniques consistently give better predictions, having a representation at all the test phases. Human predictions are also among the better techniques at two of the four test phases. We conclude that human predictions can be supported well by the use of search-based techniques and a mix of the two approaches has the potential to provide improved results.

The results of the survey based on an industrial questionnaire showed that the experts agreed on the usefulness of predicting FST (RQ1) in general for several themes, especially that the FST predictions are useful for:

- Baselining the expected effort to be spent on different test phases.
- Reducing the overall cost of testing by improving test effort on phases with high fault slippage.
- Visualizing ahead in time the expected fault-detection efficiency of test phases.

Regarding the usability of the prediction task (RQ4), 80% of the respondents are in agreement on three usability aspects:

- The techniques used for prediction need to be part of an automated tool.
- We need to know the working principles of the prediction technique to understand how is it working.

- The techniques used for prediction should be able to determine the form of relationship between inputs and outputs rather than the technique being dependent on the user for providing the form of the relationship.

The above usability aspects give indications for additional evaluation criteria that are important in addition to measuring the predictive accuracy and the goodness of fit measures. A general multi-criteria based evaluation system can thus be formulated that captures both the quantitative and the qualitative aspects of such a prediction task. Future work will also investigate ways to incorporate human judgement in the automated search mechanism.

The next chapter, Chapter 6, investigates the possibility of using the number of faults slipping from unit and function test phases to predict the fault-prone components at integration and system test phases.

Chapter 6

Prediction of fault-prone software components: A faults-slip-through approach

Originally published in Proceedings of the 17th Asia Pacific Software Engineering Conference (APSEC'10) (Best paper award)

W. Afzal

6.1 Introduction

The number of faults in a software component or in a particular release of software represents quantitative measures of software quality. A fault prediction model uses historic software quality data in the form of metrics (including software fault data) to predict the number of software faults in a component or a release [178, 209]. Automatic prediction of fault-prone components can be of immense value for a software testing team especially as we know that 20% of a software system is responsible for 80% of its errors, costs and rework [35]. Some of the benefits of using software fault prediction include: (*a*) software quality can be improved by focussing on a subset of software components. This in turn can reduce software failures and hence the maintenance costs. (*b*) Refactoring candidates can be identified for reliability enhancement [65]. (*c*) Testing activities can be better planned.

While there is a plethora of studies on software quality classification, none of them focus on identifying fault-prone components at different test levels (such as unit, function, integration and system). Secondly, due to lack of quantification of quality at test levels, all the faults are assumed to be found at the right level, which is not the case with many projects. This leads us to the concept of Faults-Slip-Through (FST) [80]. An introduction to FST is given in Section 5.1 of Chapter 5 of this thesis.

In this chapter we aim at predicting the fault-prone software components before integration and system test levels based on the FST metric. The choice of these test levels for quality enhancement is because they represent the last test levels before the software is released for customer use. Therefore, these test levels need to find as many faults as possible. In particular, we make use of number of faults slipping from unit and function test levels to predict the fault-prone components at the integration and system test levels. We essentially seek an answer to the following research question:

RQ: How can we use FST to predict fault prone software components before integration and system test and what is the resulting prediction performance?

The expectation is that answering this research question would provide valuable decision-support for the project and test managers. This decision-support relates to reduction in the number of faults slipping through to the end customer (lower maintenance and contended customers). We have used a number of classification techniques (logistic regression, C4.5, random forests, naïve Bayes, support vector machines, artificial neural networks, genetic programming and artificial immune recognition systems) to the task of classifying the quality of components. We used FST and the associated affected components' data from two large industrial projects from the telecommunication domain. The results show that FST data has the potential to be a generally useful predictor of quality of components while genetic programming show an appealing degree of classification performance.

The remainder of the chapter is organized as follows. Section 6.2 summarizes the related work. Section 6.3 describes the research context, including the variables and the data collection method. A brief on different classification techniques is given in Section 6.4. Section 6.5 gives an overview of how the performances of different techniques are evaluated. Results are given in Section 6.6 and are discussed in Section 6.7. Validity evaluation and conclusions makeup Sections 6.8 and 6.9 respectively.

6.2 Related work

There have been a number of techniques used for software quality modeling (classifying fault-proneness or predicting number of software faults) based on different sets of metrics. The applicable techniques include statistical methods, machine learning methods and mixed algorithms [67]. A number of metrics have been used as independent variables for software quality modeling and can broadly be classified into three categories [151]: (*a*) source code measures (structural measures), (*b*) measures capturing the amount of change (delta measures) and, (*c*) measures collected from meta data in the repositories (process measures). A non-exhaustive summary of software quality modeling studies appear in Section 4.2 of this thesis.

While it is clear that previous studies have focused on predicting the fault-proneness of software components, none of them quantify the quality of components at different test levels. Quantification of quality of components at different test levels promises to provide opportunities of more focussed improvements at each test level. The current study is unique from previous studies in two ways. Firstly the study aims to provide indications of fault-prone components before starting integration and system test levels. Secondly, the classification of fault-prone components is done by making use of faults-slip-through (FST) data.

6.3 Research context

The data used in this study comes from two large projects at a telecommunication company that develops mobile platforms and wireless semiconductors. The projects are aimed at developing platforms introducing new radio access technologies written using the C programming language. The average number of persons involved in these projects is approximately 250. We have data from 106 components from the two projects. Since a high percentage of components were reused in the two projects, we evaluate the fault-proneness of 106 components as if they are from a single project. We use a 10-fold cross-validation to evaluate the performance of different techniques.

The management of these projects follow the company's general project model called PROPS (PROfessional Project Steering). PROPS is based on the concepts of tollgates, milestones and check-points to manage and control project deliverables. Tollgates represent long-term business decisions while milestones are predefined events at the operating work level. The monitoring of these milestones is an important element of the project management model. The checkpoints are defined in the development process to define the work status in a process. Figure 5.4 in Chapter 5 presents an abstract level view of these concepts.

At the operative work level, the software development is structured around work packages. These works packages are defined during the project planning phase. The work packages are defined to implement change requests or a subset of a use-case, thus the definition of work packages is driven by the functionality to be developed. An essential feature of work packages is that it allows for simultaneous work on different components of the project at the same time by multiple teams. Figure 5.5 in Chapter 5 of this thesis gives an overview of how a given project is divided into work packages that affects multiple components. The division of an overall system into sub-systems in driven by design and architectural constraints.

The prediction models in this study make use of number of faults that should have been found at test levels prior to integration and system test. Since these faults were cost-effective to be found at test levels earlier than integration and system test, they are said to have slipped-through from the earlier test levels. These earlier test levels in our case are review, unit and function levels. The purpose of different test levels, as defined at our subject company, is given below:

- Review: To find faults in the feasibility of requirements, design and architecture.
- Unit: To find faults in component internal functional behavior, e.g., memory leaks.
- Function: To find faults in functional behavior involving multiple components.
- Integration: To find configuration, merge and portability faults.
- System: To find faults in system functions, performance and concurrency.

Some of these earlier test levels are composed of constituent test activities that jointly make up the higher-order test levels. Following is the division of test levels (i.e., review, unit, function, integration and system) into constituent activities at our subject company:

- Review: component design review, code review.
- Unit: Hardware development, Component test.
- Function: Function test.
- Integration: Integration of components to functions, integration test.
- System: System test, delivery test.

We collected the fault data for different components that slipped from review, unit and function test levels to the integration and system test levels. Thus we can classify the components as being either fault-prone or non-fault-prone at the integration and system test levels based on whether a single or no fault slipped through to these test levels from earlier levels.

This association of components, with the levels where the faults were to be found in them, provides an intuitive and easy way to identify fault-prone software components at different test levels. For instance, consider a fault in component A that slipped from unit level and was not captured until at integration level. Now the component A which is already fault-prone at the unit level, is more costly for quality improvement at the later integration level due to the higher cost of finding and fixing the faults at that level. But due to certain reasons (e.g., ambiguous requirements) the fault is not detected at the right level and slipped. The integration test level now has to detect both the faults that are *expected* to be found in this level and also any other faults that *slipped* from earlier levels of review, unit and function test. At this stage, any indication of fault-prone components would help plan better for integration testing. Also since integration test (and system test for that matter) represents one of the last test levels before the system is delivered to the end-users, it is critical that these last test levels have accurate knowledge of where to focus the testing effort.

Therefore we are interested in identifying those components that were fault-prone in earlier test levels of review, unit and function but the faults from these components slipped to integration and system test levels where they were eventually found. With a historical backlog of faults slipping through from review, unit and function test to integration and system test, along with the affected components, it is possible to build prediction models to predict fault-prone components for an on-going project before the commencement of integration and system test levels.

Our data set contains seven count metrics and the descriptions are given in Table 6.1. The data set contains two additional attributes that represent the dependent variables: FP-I (fault-prone at integration test level) and FP-S (fault-prone at system test level). Each one of these metrics is collected for every component separately; for example SF-CR in Table 6.1 represents the count of faults slipping from code review for each component. The data regarding the number of faults slipping to/from different test levels is readily available from an automated report generation tool at the subject company that used data from an internally developed system for fault logging.

Table 6.1: Metric descriptions.

Metric	Definition
SF-CR	No. of faults slipping from (SF) code review (CR)
SF-MDR	No. of faults slipping from (SF) module design review (MDR)
SF-R	No. of faults slipping from (SF) review (R)
SF-HD	No. of faults slipping from (SF) hardware development (HD)
SF-MT	No. of faults slipping from (SF) module test (MT)
SF-U	No. of faults slipping from (SF) unit level (U)
SF-F	No. of faults slipping from (SF) function level (F)

6.4 A brief background on the techniques

We compare a variety of techniques for the purpose of predicting fault-prone components at integration and system test. Below is a brief description of these methods while the detailed descriptions can be found in relevant references.

6.4.1 Logistic regression (LR)

Logistic regression is used when the dependent variable is dichotomous (e.g., either fault-prone or non-fault-prone). Logistic regression does not assume that the dependent variable or the error terms be normally distributed. The form of the logistic regression model is:
$log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$
where p is the probability that the fault was found in the component that slipped to either integration or system test and X_1, X_2,..., X_k are the independent variables. β_0,β_1, ..., β_k are the regression coefficients estimated using maximum likelihood. A multinomial logistic regression model with a ridge estimator, implemented as part of WEKA, was used with default parameter values.

6.4.2 C4.5

C4.5 is the most well-known algorithm in the literature for building decision trees [199]. C4.5 first creates a decision tree based on the attribute values of the available training data such that the internal nodes denote the different attributes, the branches correspond to value of a certain attribute and the leaf nodes correspond to the classification of the dependent variable. The decision tree is made recursively by identifying the attribute(s) that discriminates the various instances most clearly, i.e., having the highest information gain. Once a decision tree is made, the prediction for a new instance is done by checking the respective attributes and their values. For our experiments, standard pruning factors as in WEKA were used i.e., with a confidence factor of 0.25.

6.4.3 Random forests (RF)

Random forests is a collection of tree-structured classifiers [43]. A new instance is classified on each tree in the forest. Results of these trees are used for majority voting and the forest selects the classification having the most votes over all the trees in the forest. Each classification tree is built using a bootstrap sample of the data. The results were generated using 500 trees (the default value in random forest literature [116]).

6.4.4 Naïve Bayes (NB)

The naïve Bayes classifier is based on the Bayesian theorem. It analyses each data attribute independently and being equally important. The naïve Bayes classifier assigns an instance s_k with attribute values ($A_1 = V_1$, $A_2 = V_2$, ..., $A_m = V_m$) to class C_i with maximum $prob(C_i|(V_1, V_2, \ldots, V_m))$ for all i. The results were generated using default parameter values in WEKA.

6.4.5 Support vector machines (SVM)

The details regarding the SVM algorithm are given in Section 4.6.3 of this thesis. Using WEKA, the kernel function used was Gaussian (RBF) while other parameters had default values.

6.4.6 Artificial neural networks (ANN)

The details regarding ANN are given in Section 4.6.2 of this thesis. A three layer feed forward neural network model has been used in this study. The final ANN structure consisted of one input layer, one hidden layer and one output layer. The hidden layer consisted of five nodes while the output layer had two nodes representing each of the binary outcomes. The number of independent variables in the problem determined the number of input nodes. The sigmoid and linear transfer functions have been used for the hidden and output nodes respectively, with training time set to 500 epochs.

6.4.7 Genetic programming (GP)

The background to GP is given in Section 3.3 of this thesis. The GP programs were evaluated according to the sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$, where e_i is the actual fault count data,

$e_i^{'}$ is the estimated value of the fault count data and n is the size of the data set used to train the GP models. The control parameters that were chosen for the GP system are shown in Table 6.2.

Table 6.2: GP control parameters.

Control parameter	Value
Population size	50
Termination condition	500 generations
Function set	$\{+,-,*,/,\text{sin},\text{cos},\text{log},\text{sqrt}\}$
Tree initialization	Ramped half-and-half method
Probabilities of crossover, mutation, reproduction	0.8, 0.1, 0.1
Selection method	lexictour

6.4.8 Artificial immune recognition system (AIRS)

A brief introduction to AIRS is given in Table 5.3 of Chapter 5 of this thesis.

The WEKA plug-in for AIRS [53] has been used with the following parameters: Affinity threshold = 0.2, clonal rate = 10, hypermutation rate = 2, knn = 3, mutation rate = 0.1, stimulation value = 0.9 and total resources = 150.

6.5 Performance evaluation

We use the area under the receiver operating curve (AUC) as the performance evaluation measure for different classifiers. Area under the curve (AUC) [41] acts as a single scalar measure of expected performance and is an obvious choice for performance assessment when ROC curves for different classifiers intersect [209] or if the algorithm does not allow configuring different values of the threshold parameter. AUC, as with the ROC curve, is also a general measure of predictive performance since it separates predictive performance from class and cost distributions [209]. The AUC measures the probability that a randomly chosen fault-prone component has a higher output value than a randomly chosen non fault-prone component [98]. The value of AUC is always between 0 and 1; with a higher AUC is preferable indicating that the classifier is on average more effective in identifying fault prone components.

We also plotted the (PF, PD) pairs belonging to various classification algorithms to facilitate visualization [226] (PF denotes the probability of false alarm, i.e., proportion of non-fault prone components that are erroneously classified and PD denotes the probability of detection of fault-prone components).

6.6 Experimental results

The (PF, PD) pairs for different techniques for detecting fault-prone software components at *integration* test level are given in Table 6.3. The corresponding location of these pairs in the ROC space is shown in Figure 6.1.

Table 6.3: (PF, PD) pairs for fault prediction at integration test level.

Techniques	PF	PD	Techniques	PF	PD
GP	0.055	0.706	NB	0.673	0.921
LR	0.4	0.627	SVM	1	0
C4.5	0.64	0.94	ANN	0.545	0.784
RF	0.6	0.76	AIRS	0.473	0.706

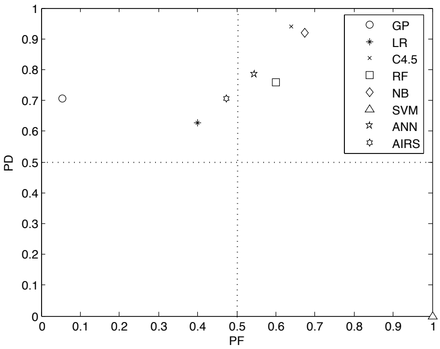


Figure 6.1: (PF, PD) pairs for different techniques for fault prediction at integration test level.

What is evident from Figure 6.1 is that three out of eight classifiers (GP, LR, AIRS) are placed in the upper left region of the ROC space, which is the region of interest for the software engineers, marked by high probability of detection (PD) and low probability of false alarm (PF). However, except GP, no technique is seen as being approximately near to the perfect classification performance of having (PF, PD) pair equal to (0, 1). GP has the (PF, PD) pair most nearer to (0, 1), in comparison with LR and AIRS, in the upper left region of the ROC space. Rest of the five classifiers (SVM, ANN, RF, C4.5, NB) are either placed in upper right or lower right regions, and hence show less impressive performance. This means that these five classifiers offer little decision-support to the software engineers in comparison with GP, LR and AIRS.

One way to quantify the distances of individual (PF, PD) pairs from the perfect classification (0, 1) is to use a distance metric, *ED* [226]:

$$ED = \sqrt{\Theta * (1 - PD)^2 + (1 - \Theta) * PF^2}$$

where Θ (ranging from 0 to 1) represents the weights assigned to PD and PF. If we assume that lower PD is more costly than a higher PF, one can assign more weight to $(1 - PD)$. Table 6.4 calculates the distance metric, ED, for the three visibly better classifiers for an arbitrary range of Θ values. The smaller the distance, i.e., the closer the point is to the perfect classification, the better the performance of the classifier [226]. GP show smaller distances in comparison with other two classifiers; a trend that is also confirmed from visualizing their (PF, PD) pairs in Figure 6.1.

Table 6.4: Distance metric, ED, values for the three better techniques for fault prediction at integration test level.

Θ	ED_GP	ED_LR	ED_AIRS
1	0.294	0.373	0.294
0.9	0.279	0.376	0.376
0.8	0.264	0.378	0.378
0.7	0.248	0.381	0.381
0.6	0.230	0.384	0.384

Table 6.5 shows the AUC measures for different techniques for predicting fault-prone components at integration test level. GP shows the highest AUC value followed by C4.5, ANN and AIRS for fault prediction at the integration test level. SVM has the lowest AUC value and is also reflected by its (PF, PD) pair being in the lower right corner of the ROC space. Also what is evident from these AUC values is that apart from GP and SVM (having best and the worst AUC values), the rest of the techniques show small differences.

Table 6.5: AUC measures for different techniques at integration test level.

GP	LR	C4.5	RF	NB	SVM	ANN	AIRS
0.853	0.614	0.652	0.58	0.624	0.50	0.619	0.617

The (PF, PD) pairs for different techniques for detecting fault-prone components at *system* test level are given in Table 6.6.

Table 6.6: (PF, PD) pairs for fault prediction at system test level.

Techniques	PF	PD	Techniques	PF	PD
GP	0.10	0.73	NB	0.16	0.29
LR	0.605	0.81	SVM	1	1
C4.5	0.87	0.87	ANN	0.76	0.88
RF	0.737	0.720	AIRS	0.53	0.647

The corresponding location of these pairs in the ROC space is shown in Figure 6.2. This time only a single technique (GP) has its (PF, PD) pair in the preferred upper left region of the ROC space. Six of the techniques (AIRS, LR, RF, ANN, C4.5, SVM) resulted in being placed at top right region of the ROC space while NB got a place in lower left. For the second time, SVM is not able to provide useful results, with a high probability of false alarms. LR, C4.5, RF and ANN are closely placed in the upper right region with small differences in their (PF, PD) values, though AIRS and LR have their (PF, PD) pairs closer to the preferred region.

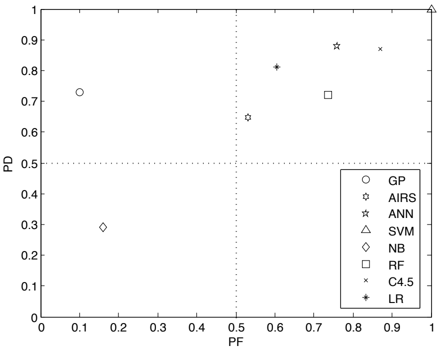


Figure 6.2: (PF, PD) pairs for different techniques for fault prediction at system test level.

Table 6.7 calculates the distance metric, ED, for the three visibly better classifiers for an arbitrary range of Θ values.

Table 6.7: Distance metric, ED, values for the three better techniques for fault prediction at system test level.

Θ	ED_GP	ED_LR	ED_AIRS
1	0.27	0.19	0.353
0.9	0.258	0.263	0.374
0.8	0.246	0.319	0.395
0.7	0.232	0.367	0.414
0.6	0.218	0.410	0.432

It shows that GP has smaller distances on majority of Θ values in comparison with LR and AIRS, while LR's ED values are better than those of AIRS. This trend is also confirmed from visualizing the respective techniques' (PF, PD) pairs in Figure 6.2.

Table 6.8 shows the AUC measures for different techniques for predicting fault-prone components at system test level. GP has the highest AUC value, followed-up by LR. Rest of the techniques showed small differences in their AUC values.

Table 6.8: AUC measures for different techniques at system test level.

GP	LR	C4.5	RF	NB	SVM	ANN	AIRS
0.78	0.602	0.5	0.49	0.568	0.50	0.56	0.56

Now that we have the AUC values for different techniques for predicting fault-prone components at integration and system test levels, we can use a two-sample t-test to verify if the differences in AUC values among different pairs of techniques are significant or not. The results of applying the two sample t-test appear in Table 6.9 where $h = 1$ indicates a rejection of the null hypothesis at 5% significance level of the two samples having equal means, while $h = 0$ indicates a failure to reject the null hypothesis at 5% significance level.

The results show that apart from seven combinations, highlighted in bold in Table 6.9, there are no significant differences between the AUC values of all other combination of classifiers. It is, however, interesting to note that there are significant differences between the AUC values of GP and all other classifiers except for one (C4.5). This shows that GP has an edge in terms of identifying fault-prone components at integration and system test levels in comparison with majority of the techniques.

Table 6.9: Two-sample *t*-test results for differences in AUC measures for different techniques.

Techniques	*p* value	h	Techniques	*p* value	h
GP:LR	**0.030**	**1**	C4.5:NB	0.828	0
GP:C4.5	0.104	0	C4.5:SVM	0.423	0
GP:RF	**0.040**	**1**	C4.5:ANN	0.884	0
GP:NB	**0.041**	**1**	C4.5:AIRS	0.892	0
GP:SVM	**0.013**	**1**	RF:NB	0.369	0
GP:ANN	**0.040**	**1**	RF:SVM	0.518	0
GP:AIRS	**0.039**	**1**	RF:ANN	0.418	0
LR:C4.5	0.715	0	RF:AIRS	0.421	0
LR:RF	0.250	0	NB:SVM	0.076	0
LR:NB	0.716	0	NB:ANN	0.888	0
LR:SVM	**0.003**	**1**	NB:AIRS	0.868	0
LR:ANN	0.601	0	SVM:ANN	0.094	0
LR:AIRS	0.572	0	SVM:AIRS	0.090	0
C4.5:RF	0.688	0	ANN:AIRS	0.983	0

6.7 Discussion

In this chapter, we evaluated the use of several techniques for predicting fault-prone components at integration and system test levels using faults-slip-through data from two industrial projects. The results of this study have multiple important dimensions. First, this chapter demonstrates the use of faults-slip-through metric as a potential predictor of fault-proneness. While previous studies have focussed on structural measures, change measures and process measures (Section 6.2) as predictors of fault proneness, this study shows that the use of number of faults slipping through to/from various test levels are able to provide good results for finding fault-prone components at integration and system test levels. At integration and system test levels, GP was able to give impressive AUC values (0.7 or more – with the (PF, PD) pairs in the preferred region of the ROC space). For rest of the techniques, five of them had their AUC values greater than 0.6 at integration test level while one of them showed a AUC value greater than 0.6 at system test level. Though still moderate (for techniques other than GP), these AUC values give an early indication that faults-slip-through measures have the discriminative power to classify components as either fault-prone or non-fault-prone at integration and system test levels. This result strengthens Arisholm's et al. results [26] where simple process measures significantly improved the prediction models. For future work, it would be interesting to investigate the combination of faults-slip-through metric with structural, change and other process measures for effectively finding fault-prone components.

Second, GP has shown promising results for predicting fault-proneness at both integration and system test levels. This quantitatively better performance of GP adds to the other qualitative features of using such a non-parametric search-based technique for fault prediction. These additional feature include *independence* from data distribution assumptions and *automatic* evolution of model and associated coefficients based on historical data. One additional advantage of using GP is the *comprehensibility* of resulting models which can lead to an insight of the significant predictor variables and important rules. This result adds to the current body of knowledge that explores GP as a possible tool for predictive studies in software engineering [5].

Third, previous studies on fault-proneness classified components irrespective of the different test levels. While such studies are useful, we might run into a risk of investing more effort in improving the quality of a component than is cost-effective at a certain test level. The quantification of quality of components at different test levels, coupled with the use of FST measures, entail an additional benefit that components can be selected for quality enhancement keeping in view the cost-effectiveness. Thus the fundamental hypothesis underlying the work in this study is that an efficient test process verifies each product aspect at a test level where it is easiest to test and the faults are cheapest to fix; therefore the identification of fault-prone components at specific test levels is a step in that direction. We were assisted in this endeavor to an extent by the segregation of different test levels at our subject organization.

While AUC is a generally useful measure of classification performance, however, we need additional ways to visualize the performance of different techniques. The plotting of (PF, PD) pairs in the ROC space is one way of achieving it. As clear from the two plots in Figures 6.1 and 6.2, the location of (PF, PD) pairs in the ROC space can quickly show the trade-off in PF and PD values of competing techniques whereby one or more techniques might be preferred. This additional way to visualize performance is important from the viewpoint of practical use. A software manager who intends to apply these techniques would be mainly interested in correct detection of fault-prone components, i.e., PD [324]. This is because the cost of delivering fault-prone components to the end-customers is much more than the cost of testing one component too many. This unequal cost of misclassification is the reason why software managers would prefer a trade-off between PF and PD. Using a distance metric like ED and visualizing the (PF, PD) pairs in the ROC space provides that flexibility to the software manager.

6.8 Empirical validity evaluation

Conclusion validity refers to the statistically significant relationship between the treatment (independent variable) and the outcome (dependent variable). We tested for any significant differences between the AUC values of different techniques using a two-sample t-test at 5% significance level, which is a commonly used significance level for hypothesis testing. The choice of selecting a parametric test was based on its greater power to identify differences and robustness to smaller departures from the normality assumption. *Internal validity* refers to a causal relationship between treatment and outcome. Out of the many metrics we could collect, we only selected those based on the FST metric since our goal was to evaluate the use of such metrics for quality prediction. We used a 10-fold cross-validation as a resampling method which has been found to give low bias and low variance [194]. A potential threat to internal validity is that the faults-slip-through data did not consider the severity level of faults, rather treated all faults equally. *Construct validity* is concerned with the relationship between the theory and application. The use of AUC as the performance measure is motivated by the fact that it is a general measure of predictive performance, while the plot of (PF, PD) pairs provides an intuitive visualization tool. *External validity* is concerned with generalization of results outside the scope of the study. The data used in the study comes from two industrial projects from the telecommunication domain and thus represents real-life use. However, more replications of this study would help generalize the results beyond the specific environment.

6.9 Conclusion and future work

This chapter evaluated the use of faults-slip-through data as a potential predictor of fault-proneness at integration and system test levels for data gathered from two industrial projects. A variety of classification algorithms were applied, including a standard statistical technique for classification (logistic regression), tree-structured classifiers (C4.5 and random forests), a Bayesian technique (Naïve Bayes), machine-learning techniques (support vector machines and back-propagation artificial neural networks) and search-based techniques (genetic programming and artificial immune recognition systems). The performance of these classifiers was assessed using AUC and location of (PF, PD) pairs in the ROC space. The results of this study concluded that faults-slip-through data has the potential to be a generally useful predictor of fault-proneness at integration and system test levels. As for the different classifiers, GP performed more consistently across both integration and system test levels in terms of AUC and location of (PF, PD) pairs in the ROC space. The visualization of (PF, PD) pairs in the ROC space provides another opportunity for the test team to assess a classifier performance with respect to the perfect (PF, PD) pair of (0, 1). A distance metric can then be calculated, with different weights assigned to represent the misclassification costs of PF and PD, to select a classifier most suited for the project.

Based on this study, some interesting future work can be undertaken. Firstly, it would be interesting to compare the FST metric with other commonly used predictors of fault proneness to quantify any differences. Secondly, the performance of FST as an effective predictor of fault-proneness needs to be assessed for a segregation of faults based on severity levels. Lastly, one can think of a probabilistic model on how likely different fault counts or slips in earlier phases are for predicting fault-proneness in later phases.

The next chapter, Chapter 7, answers our first of the two methodological investigation questions. Chapter 7 seeks to investigate the potential impact of resampling methods on software quality classification.

Chapter 7

Resampling methods in software quality classification: A comparison using genetic programming

Submitted to the International Journal of Software Engineering and Knowledge Engineering, special issue on: Emerging Synergies of Artificial Intelligence and Software Engineering

W. Afzal, R. Torkar and R. Feldt

7.1 Introduction

Different dependent variables of interest have been the target of predictive studies in software engineering. Software quality classification is one such domain which concerns classifying software components as either fault-prone (fp) or non-fault prone (nfp). A fault prone component is one in which the number of faults are higher than a selected threshold. Such a classification of software components potentially has an effect on overall software quality since fault-prone components are candidates for further reliability enhancement. Supervised learning algorithms from machine learning literature represent one of the relatively newer approaches for software quality classification whereby an induction algorithm builds a classifier from a given data set. Examples of such studies include applications of artificial neural networks, e.g., [167], classification and regression trees CART, e.g., [171], support vector machines, e.g., [315] and evolutionary computation, e.g., [6, 215].

With the availability of numerous techniques for constructing classification models, an important task in quality classification is appropriate model selection and evaluation. There are several key questions to answer in achieving this task, e.g.,

1. What resampling method to use?

2. What prediction accuracy measure to use?

3. What statistical tests to use to compare the results?

While each one of these questions are important, the focus of this study is to answer the choice of a resampling method to use. Specifically, we attempt to investigate which resampling method performs better while using genetic programming as a software classification technique.

It is common in machine learning that a portion of a data set is used to test the performance of the trained classifier. With limited data, different resampling methods are used to assess a model's generalizability. The choice of a resampling method is an important element in an overall model selection procedure that has attracted little investigation. With lack of convergence across various software classification models, the researchers have highlighted the need of greater use of public data sets, appropriate accuracy indicators and statistical testing procedures; but the choice of a resampling method is surprisingly less emphasized in the past. However increasing concern to investigate the choice of resampling methods is now being raised. Myrtveit et al. [250] and Lessmann et al. [209] highlight the need to examine the influence of resampling methods and to quantify possible differences. Kitchenham and Mendes [190] point at the importance of explicitly stating the resampling method chosen:

> "Another important issue is whether to compare with predictions based on the entire data set or predictions based on dividing the data into training and testing data sets. Most researchers agree that the latter technique is better, but if we use anything other than a simple leave-one-out procedures results are not auditable unless the specific data set partitions are defined."

Further highlighting the need of multiple training sets, Kirsopp and Shepperd [186] came to the conclusion:

> "The major conclusion of this paper is, however, that it is dangerous to make inferences concerning the accuracy of prediction systems based on a small number of sampled training sets."

Kitchenham et al. [191] also highlight that a variety of resampling techniques used by different studies is one of the reasons that impedes a formal meta-analysis of the primary study results:

> "Some studies used independent holdout samples; others used different types of cross validation (e.g., 3-fold, 20-fold, leave-one-out cross validation) ... These differences made it impossible to perform any formal meta-analysis of the primary study results."

A recent systematic review comparing genetic programming (GP) with other methods of predictive studies [5] indicate that for studies applying GP for software fault prediction and software reliability growth modeling, it is not always clear which resampling method is used (Table 7.1). The third column in Table 7.1 indicate that four out of eight studies did not mention the resampling method used while three out of eight studies used hold-out validation. This shows that while use of resampling methods is an unsettled matter in predictive studies in software engineering in general, it requires even more investigation when using GP as a prediction technique.

Table 7.1: Data set characteristics for primary studies on GP application for software fault prediction and reliability growth. All studies included the use of industrial data. (?) indicates absence of information.

Study	No. of data sets	Sampling of training and testing sets	Data sets public or private
[160]	1	?	Public
[161]	1	?	Public
[322]	1	10-fold cross-validation	Public
[355]	1	?	Private
[356]	1	?	Private
[6]	3	First $\frac{2}{3}$ of the data set for training and the rest for testing	Private
[73]	2	First $\frac{2}{3}$ of the data set for training and the rest for testing	Public & Private
[262]	1	First $\frac{2}{3}$ of the data set for training and the rest for testing	Public

We therefore seek an answer to the following research question in this chapter:

RQ: How do different resampling methods compare with respect to predicting fault-prone software components using GP?

The affect of resampling methods on GP evolution has sporadically been discussed in research but not in a manner as in this study. Ross [283] studied the effects of randomly sampled training data on program evolution in GP. The study concluded that GP performance is better when the samples are more representative of the target behavior. The study did not empirically evaluate different resampling methods but highlighted a need of doing so:

> “The effects of re-sampling are not as clear, and further work is required to study the relationship between re-sampling rates and GP performance.”

In this study, we use GP as a software quality classification approach and evaluate the influence of different resampling methods on the outcome of software quality classification. We present an extensive comparison between five common resampling methods: hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping using five different publicly available data sets.

The rest of the chapter is organized as follows: Section 7.2 presents relevant related studies. Section 7.3 presents the study design including an introduction to the different resampling methods used, an introduction to genetic programming and symbolic regression, the data sets used, performance estimation of classification accuracy and the experimental setup. The results are presented in Section 7.4, and discussed in Section 7.5. Validity issues make up Section 7.6 while conclusions appear in Section 7.7.

7.2 Related work

Few comparisons of standard resampling methods have been performed in software engineering. Mittas and Angelis [238] used permutation tests and bootstrap to construct confidence intervals for the difference in accuracy measures for software cost prediction using regression and estimation by analogy. The emphasis of their study was not to find the best model but rather to recommend a systematic comparison of models using statistical hypothesis testing. Kirsopp and Shepperd [186] analyzed the influence of number of training sets for software effort prediction using case-based prediction on two data sets. They evaluated the hold-out procedure and demonstrated that results may be misleading unless at least 5 different training sets, and preferable more than 20, are used. Green and Ohlsson [114] used artificial neural network ensembles to compare 5×5 fold cross-validation, 25-fold bootstrap and 25-fold hold-out using three cut-offs (0.25, 0.50 and 0.75). They showed that $5{\times}5$ fold cross-validation and hold-out with cut-offs 0.25 and 0.50 are the best resampling strategies for estimating the true performance of ANN ensembles. Kohavi [194] experimented with $C4.5$

and Naive-Bayesian classifier using six industrial data sets to compare 0.632 bootstrap and k-fold cross-validation with different values of k. They concluded that 10-fold stratified cross-validation is the best method for the data sets used. A study by Schaffer [287] concluded that on average 10-fold cross-validation strategy outperforms $C4.5$ decision-trees and back-propagation neural networks. Molinaro et al. [242] used four classification algorithms (linear discriminant analysis, diagonal discriminant analysis, nearest neighbor and classification and regression trees (CART)) to compare different resampling methods. Among several conclusions, one of them was that leave-one-out cross-validation and 10-fold cross-validation had the smallest bias for diagonal discriminant analysis, nearest neighbor, CART and linear discriminant analysis.

While other references to resampling methods may be found in literature, we have focussed above on more recent ones and their use in comparative studies.

7.3 Study design

In this section we present an introduction to the different resampling methods used, an introduction to genetic programming and symbolic regression, the data sets used, performance estimation of classification accuracy and the experimental setup.

7.3.1 Resampling methods

Resampling is an important concept in inferential statistics. It is used to draw a large number of samples from the original one and thus to reach an approximation of the underlying theoretical distribution. It is based on repeated sampling within the same data set [351].

Intuitively the most accurate way to establish an approximation of the underlying theoretical distribution is to examine the values for every member of the population but this might not be possible due to high costs and high time-consumption [111]. That is why we need accurate methods to use sample statistics to estimate population parameters. Parametric statistics relies on the properties of the central limit theorem to generalize sample statistics. Problems associated with most data sets threaten the assumptions of these approaches. Conventional non-parametric alternatives offer freedom from distribution assumptions but more complex experimental designs do not have non-parametric equivalents and there is always a concern for loss of statistical power. Resampling methods are a new dimension of distribution-free methods that address many of the limitations of conventional parametric and non-parametric methods [271]. Resampling methods are particularly useful in the absence of large, independent test sets with normal distributions [32].

Resampling is especially important for the validity of software engineering predictive studies since software engineering data sets are scarce and data limited. This has to do with difficulties in getting large data sets due to the data being confidential or where the data simply is too rudimentary in its nature [7]. The below mentioned points highlight the benefits of using resampling methods [284, 351]:

1. Resampling methods do not make any assumptions about data distribution.
2. One can analyze any statistic.
3. Resampling methods are less complex and do not require sophisticated mathematical background.
4. There is no specific sample size restriction.
5. Resampling methods provide a way to achieve internal replication [308].

The resampling methods divide the data into learning set and a test set. This division is crucial since performance of a particular modeling technique on the learning set can not be a good indicator of performance on an independent test set [341]. The learning set is the one that is used to train the models using different techniques. Sometimes it is required to split the learning data further into two sets: training set and the validation set. The training set is used for the models to learn from the data and then the validation set is used to optimize the parameters of the learned models. The test set is used to evaluate the trained models on one or more accuracy measures. We will solely examine the partitioning of data into training and test sets for the purpose of delimiting ourselves to reach an early evaluation of the resampling methods.

We examine the two most common resampling methods: cross-validation and bootstrapping. We also compare the split-sample or the hold-out method which acts as a baseline and an obvious split choice [341]. We briefly describe these methods below and refer the reader to a more detailed discussion in [93, 94, 304].

Cross-validation

Cross-validation (CV) is widely used in regression and classification problems to obtain nearly unbiased estimators of generalization errors. The inherent idea is to train and test the model on separate subsets of data to get model's prediction strength as a function of a CV error curve. Cross-validation (CV) consists of [93]:

1. deleting the points x_i from the data set one at a time;
2. recalculating the prediction rule on the basis of the remaining $n-1$ points;

3. seeing how well the recalculated rule predicts deleted point, and;
4. averaging these predictions over all n deletions of an x_i.

Cross-validation is primarily used to reduce over-fitting, which refers to a model's poor performance on the test set or lack of generalization. There are generally three types of cross-validation: repeated random subsampling, k-fold cross validation and leave-one-out cross-validation.

Repeated random sub-sampling validation (Monte-Carlo cross-validation) This method randomly divides (without replacement) the original sample into training and test sets numerous times. The results of the trained model are evaluated using the test set for every sub-sample (Figure 7.1a [119]).

A disadvantage of this approach is that some observations may never be selected in the test sets whereas others may be selected more than once. Therefore the data may not be optimally used in every split of repeated random sub-sampling.

In this study we randomly split the sample into learning and test set 10 times. In each split one third of the sample was used as a test set while the remaining two-third was used as a training set. For each split we ran the GP algorithm 10 times and picked the GP solution giving the least classification error. We calculated the sample statistics – AUC, PF and PD (discussed in Section 7.3.4) for each of the 10 splits and present the average.

***k*-fold cross-validation (Rotation estimation)** This method splits the data into k subsets (or folds) and each one of these k subsets is used as a testing set to evaluate the models' performance by using $k-1$ subsets as training set. The cross validation process is repeated k times. The evaluation measure (e.g., the error estimate) for each of these folds is averaged to reach an overall measure (or the error estimate). K-fold cross-validation has the advantage that all values in the data set are eventually used for both training and testing. A common choice for k-fold cross-validation is $k = 10$; thus called as 10-fold cross-validation [341] (Figure 7.1b). We used 10-fold cross-validation in this study.

For each split, we ran the GP algorithm 10 times and picked the GP solution giving the least classification error. We calculated the sample statistics – AUC, PF and PD (discussed in Section 7.3.4) for each of the 10 splits and present the average.

Leave-one-out cross-validation (LOOCV) In LOOCV, each observation in the sample is used once in the test set. Using a single observation as the test set allows the model to be trained on the remaining observations. This is the extreme case of k-fold cross-validation where k being equal to the number of observations in the test set (Figure 7.1c). For large samples LOOCV might not be favorable due to high computational times but the advantage is that it uses the greatest possible amount of data for training [341].

For each split, the best GP solution was picked and sample statistics (AUC, PF and PD) were calculated. The average of these statistics in then presented.

bootstrapping

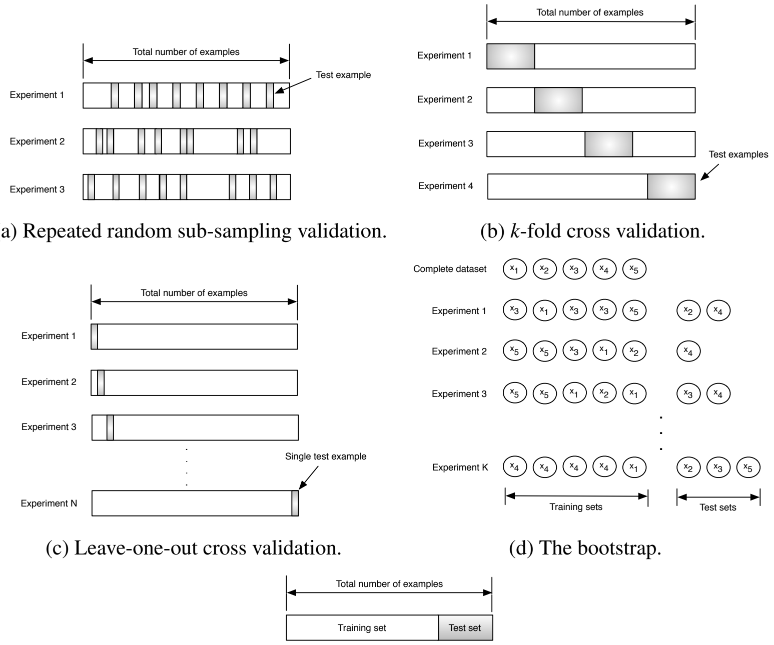
bootstrapping is a resampling method *with* replacement. For a data set of size n, a bootstrap sample is created by randomly selecting (with replacement) n examples and this set is then used for training. Because some elements in the training set will be repeated, the testing set consists of the instances in the original data set that have not been picked in the training set. The process is repeated for a specified number of bootstrap samples or folds (Figure 7.1d). There are several variations of the bootstrap resampling method but we will only use the non-parametric bootstrap method, which makes no assumption about the form of the population's distribution. The non-parametric bootstrap work as follows [327]:

- Collect the data set of n samples $\{x_1,\ldots,x_n\}$.
- Create B bootstrap samples $\{x_1^*,\ldots,x_n^*\}$ where each x_i^* is a random sample with replacement from $\{x_1,\ldots,x_n\}$.
- For each bootstrap sample $\{x_1^*,\ldots,x_n^*\}$ calculate the required statistic θ. The distribution of these B estimates of θ represents the bootstrap estimate of uncertainty about the true value of θ.

The required statistics in our case are the AUC, PF and PD measures (discussed in Section 7.3.4). We repeated the bootstrap method 10 times for each data set while running the GP algorithm 10 times for each bootstrap sample. We selected the GP solution giving the minimum error rate as a result of 10 runs of GP algorithm for each bootstrap sample and calculated the required statistics. The final AUC, PF and PD statistics were taken as an average from each of the bootstrap samples.

Hold-out validation

Hold-out validation involves splitting the data set into a single partition consisting of a training set and a test set (Figure 7.1e). The commonly used practice is to hold-out one-third of the data for testing and to use two-thirds for training [341]; we use the same in



(a) Repeated random sub-sampling validation.

(b) *k*-fold cross validation.

(c) Leave-one-out cross validation.

(d) The bootstrap.

(e) The holdout method.

Figure 7.1: The resampling methods used for comparison.

this study. The advantage of this method is its simplicity and ease of computation but it wastes data that could have been used for improving the classifier.

7.3.2 Genetic programming (GP) and symbolic regression application of GP

An introduction to GP and symbolic regression application of GP is given in Section 3.3 of this thesis.

The GP parameters used for this study are shown in Table 7.2. The GP programs were evaluated according to the sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} \mid e_i - e_i^* \mid$, where e_i is the actual fault count

Table 7.2: GP control parameters.

Control Parameter	Value
Population size	30
Number of generations	100
Termination condition	100 generations
Function set	$\{+, -, *, sin, cos, log\}$
Terminal set	$\{x\}$
Tree initialization	ramped half-and-half
Initial maximum number of nodes	28
Maximum number of nodes after genetic operations	512
Genetic operators	crossover, mutation, reproduction
Probabilities of crossover, mutation, reproduction	0.8, 0.1, 0.1
Selection method	lexictour
Elitism	replace

data, e_i^* is the estimated value of the fault count data and n is the size of the data set used to train the GP models. The selection method used is lexicographic parsimony pressure tournament [221] (short named as *lexictour*) in which the best individuals are selected from a random number of individuals. If two individuals are equally fit, the tree with fewer nodes is chosen as the best. For a new population, the parents and offsprings are prioritized for survival according to elitism. The elitism level specifies the members of the new population, to be selected from the current population and the newly generated individuals. The elitism level used in this study is *replace*, in which children replace the parent population having received higher priority of survival, even if they are worse than their parents. GPLAB, the GP toolbox for MATLAB is used for running the GP algorithm [292].

7.3.3 Public domain data sets

The data sets used in this study are taken from the PROMISE data repository [37] which is a collection of data sets freely available for performing predictive studies in software engineering. Specifically we make use of 5 data sets from the PROMISE repository namely AR6, AR1, PC1_req, JM1_req and CM1_req.

AR6 and AR1 data sets contain fault-proneness data from two embedded software for white-goods products written in the C programming language. AR6 consists of 101 components while AR1 contains data from 121 components. The collected metrics for AR6 are McCabe, Halstead and LoC measures while for AR1 several function level static code attributes have been collected. More information on these data sets is available at [319, 320].

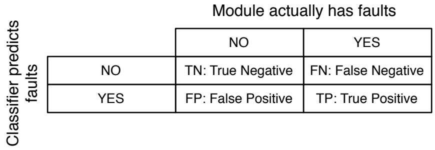
The remaining data sets (PC1_req, JM1_req and CM1_req) come from the NASA metrics data program (MDP) data repository [219]. These data sets contain different requirement metrics. Since not all the requirements were linked to the components, therefore, subsets of the original sets are used which are available at the PROMISE repository [149] and have been used in a study by Jiang et al. [148]. The data sets PC1_req, JM1_req and CM1_req relate to NASA's earth orbiting satellite system, real-time ground system and spacecraft instrument respectively. Table 7.3 provides further information regarding all data sets used in this study.

Table 7.3: Information on data sets used in the study.

Data set	No. of records	% with faults	Language	Domain	No. of predictor variables
AR6	101	14.85	C	White-goods embedded software	29
AR1	121	6.61	C	White-goods embedded software	29
PC1_req	320	33.43	C	Earth orbiting satellite system	8
JM1_req	37	45.94	C	Real-time ground system	8
CM1_req	89	77.53	C	Spacecraft instrument	8

7.3.4 Performance estimation of classification accuracy

We restrict ourselves to evaluate the performance of *binary* classifiers which categorizes instances or software components as being either fault-prone (fp) or non-fault prone (nfp). We are interested in predicting whether or not a component contains any faults, rather than the total number of faults. A common assessment procedure for binary classifiers is to count the number of correctly predicted components over hold-out (test set) data. A fault prediction sheet [234], as in Figure 7.2, is commonly used.



Module actually has faults

Classifier predicts faults		NO	YES
	NO	TN: True Negative	FN: False Negative
	YES	FP: False Positive	TP: True Positive

Figure 7.2: The fault prediction sheet (confusion matrix).

Based on the different possibilities in the fault prediction sheet various measures are typically derived. El-Emam et al. [95] have derived a number of measures based on this; the most common ones being rate of faulty component detection (or probability of detection (PD) or specificity), overall prediction accuracy (acc), probability of false alarm (PF or recall) and precision (prec). However the measure of overall accuracy acc has been criticized as being misleading since it ignores the data distribution and cost information [226]. The other measures of PD, PF and prec also reveal only one aspect of the prediction models at a time; thus their use introduces bias in performance assessment. Use of these measures also complicate comparisons and model selection since there is always a trade-off between three measures, e.g., one model might exhibit a high PD but lower prec [226].

A receiver operating characteristic (ROC) curve [98] and the area under a ROC curve (AUC) [126] have been shown to be more statistically consistent and discriminating than predictive accuracy, acc [212]. The ROC curve is also a more general way, than numerical indices, to measure a classifier's performance [350]. A ROC curve provides an intuitive way to compare the classification performances of different techniques. ROC is a plot of the trade-off between the ability of the classifier to correctly detect fault-prone components (PD) and the number of non-fault prone components that are incorrectly classified (PF) across all possible experimental threshold settings [150, 226]. In short the (PF, PD) pairs generated by adjusting the algorithms threshold settings forms an ROC curve. A typical ROC curve is shown in Figure 7.3.

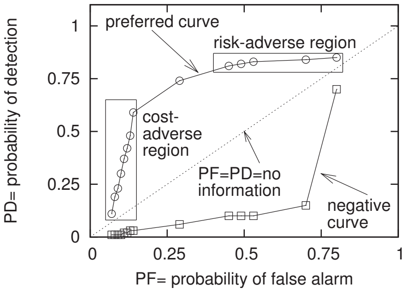


Figure 7.3: A typical ROC curve.

This concave curve has the probability of detection (PD) on *y*-axis while the *x*-axis shows the probability of false alarms (PF). The start and end points for the ROC curve are $(0,0)$ to $(1,1)$, respectively. The software engineers need to identify the points on the ROC curve that suits their risks and budgets for the project [148]. A straight line from $(0,0)$ to $(1,1)$ offers no information while the point $(\text{PF} = 0, \text{PD} = 1)$ is the ideal point on the ROC curve. A negative curve bends away from the ideal point while a preferred curve bends up towards the ideal point. As such, if we can divide the ROC space into four regions as shown in Figure 7.4, the only region with practical value for software engineers is region *A* with acceptable PD and PF values. The regions *B*, *C* and *D* represent poor classification performance and hence are of little to no interest to software engineers [226].

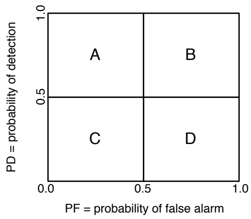

PD = probability of detection
1.0
0.5
A
B
C
D
0.0
0.5
1.0
PF = probability of false alarm

Figure 7.4: Four regions in the ROC space.

Area under the curve (AUC) [41] acts as a single scalar measure of expected performance and is an obvious choice for performance assessment when ROC curves for different classifiers intersect [209] or if the algorithm does not allow configuring different values of the threshold parameter. AUC, as with the ROC curve, is also a general measure of predictive performance since it separates predictive performance from class and cost distributions [209]. The AUC measures the probability that a randomly chosen fp component has a higher output value than a randomly chosen nfp component [98]. The value of AUC is always between 0 and 1; with a higher AUC indicating that the classifier is on average more to the upper left region *A* in Figure 7.4.

We use AUC and the location of (PF, PD) pairs in the ROC space as measures of classification performance for the different resampling methods.

7.3.5 Experimental setup

For each sample of an individual data set for each resampling method (except for LOOCV), the GP algorithm is run for 10 times (GP being a stochastic algorithm). The best GP individual from the 10 runs of the algorithm is chosen for each sample of an individual data set. The sample statistics (AUC, PF, PD) were calculated for each of these best GP individuals and then averaged.

For LOOCV, running GP algorithm for 10 times for each sample of an individual data set was not feasible due to high computational times, therefore the GP was ran once for each leave-one out sample of a particular data set. The sample statistics (AUC, PF, PD) from each leave-one-out samples were then averaged.

Figure 7.5 further illustrates the experimental procedure that was used for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation, non-parametric bootstrap resampling methods. Note that for hold-out validation, each data set is randomly split into a training set (2/3 of the data) and a testing set (1/3 of the data) as is discussed in Subsection 7.3.1.

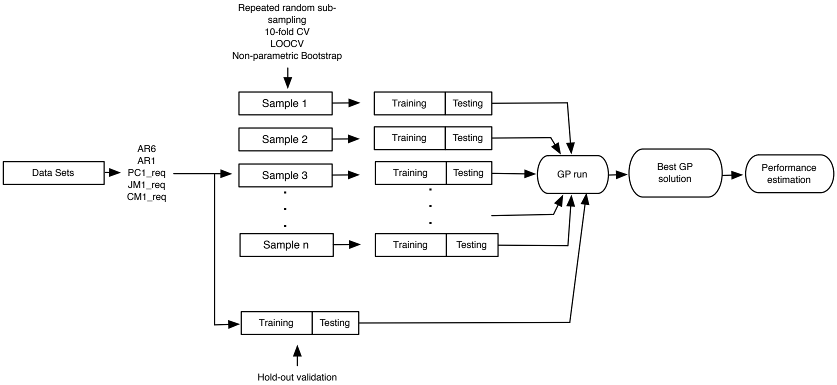


Figure 7.5: The experimental procedure.

7.4 Results

In this section we present the results of the empirical comparison in terms of (PF, PD) pair data in the ROC space and the AUC. All the results are based on the AUC, PF and PD values that represent the average over all the sub-samples (except for the hold-out validation where we have a single split of data).

7.4.1 (PF, PD) in the ROC space

AR6 data set

Table 7.4 shows the (PF, PD) pairs for the AR6 data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figure 7.6. For all the resampling methods the (PF, PD) pairs are in the region *C* of the ROC space but hold-out and bootstrap resampling methods tend to have comparatively higher PD and lower PF values which is desirable.

Table 7.4: (PF, PD) pair data for the AR6 data set.

	PD	PF
Hold-out validation	0.33	0.06
Repeated random sub-sampling validation	0.10	0.40
10-fold cross-validation	0	0.01
Leave-one-out cross-validation	0.13	0.01
bootstrapping	0.16	0.04

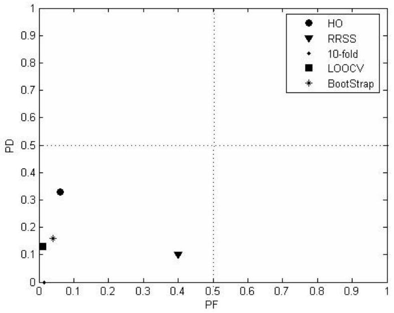


Figure 7.6: (PF, PD) pair data for the AR6 data set in the ROC space. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

AR1 data set

Table 7.5 shows the (PF, PD) pairs for the AR1 data set for each of the resampling methods. The corresponding location of these pairs in the ROC space is shown in Figure 7.7. Again as with data set AR6, all the resampling methods have (PF, PD) pairs in the region *C* while bootstrap tends to have only a slightly better PF and PD values.

Table 7.5: (PF, PD) pair data for the AR1 data set.

	PD	PF
Hold-out validation	0	0
Repeated random sub-sampling validation	0	0.05
10-fold cross-validation	0	0
Leave-one-out cross-validation	0	0
bootstrapping	0.07	0.08

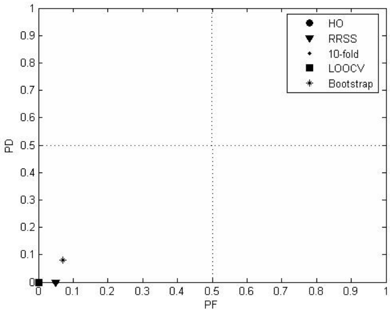


Figure 7.7: (PF, PD) pair data for the AR1 data set in the ROC space. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

PC1_req data set

Table 7.6 shows the (PF, PD) pairs for the PC1_req data set for each of the resampling methods.

Table 7.6: (PF, PD) pair data for the PC1_req data set.

	PD	PF
Hold-out validation	0	0
Repeated random sub-sampling validation	0.21	0.35
10-fold cross-validation	0.25	0.16
Leave-one-out cross-validation	0.39	0.24
bootstrapping	0.62	0

The corresponding location of these pairs in the ROC space is shown in Figure 7.8. The only method not in region *C* is bootstrap with (PF, PD) pair of (0, 0.62) that places it in region *A*.

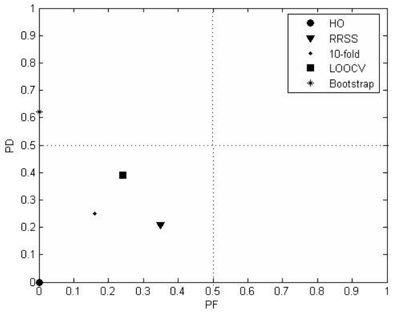


Figure 7.8: (PF, PD) pair data for the PC1_req data set in the ROC space. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

JM1_req data set

Table 7.7 shows the (PF, PD) pairs for the JM1_req data set for each of the resampling methods.

Table 7.7: (PF, PD) pair data for the JM1_req data set.

	PD	PF
Hold-out validation	1	1
Repeated random sub-sampling validation	0.54	0.46
10-fold cross-validation	0.47	0.47
Leave-one-out cross-validation	0.50	0.43
bootstrapping	0.58	0.58

The corresponding location of these pairs in the ROC space is shown in Figure 7.9. This time we see a wider spread of (PF, PD) pairs in the ROC space with repeated random sub-sampling validation having the (PF, PD) pair located in region *A*. Both hold-out validation and bootstrapping have high (PF, PD) values and are thus consequently placed in region *B*.

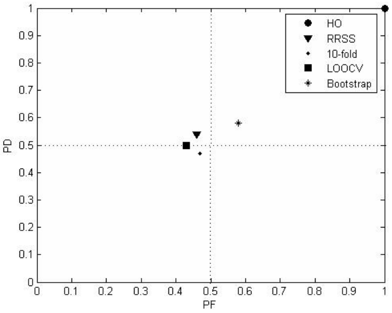


Figure 7.9: (PF, PD) pair data for the JM1_req data set in the ROC space. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

CM1_req data set

Table 7.8 shows the (PF, PD) pairs for the CM1_req data set for each of the resampling methods.

Table 7.8: (PF, PD) pair data for the CM1_req data set.

	PD	PF
Hold-out validation	1	1
Repeated random sub-sampling validation	1	1
10-fold cross-validation	1	0.33
Leave-one-out cross-validation	0.54	0.28
bootstrapping	0.96	1

The corresponding location of these pairs in the ROC space is shown in Figure 7.10. We see here the concentration of (PF, PD) pairs within two regions of the ROC space. For 10-fold cross-validation and leave-one-out cross-validation, the (PF, PD) pairs lie in region *A* while for rest of the resampling methods the (PF, PD) pairs are in region *B*.

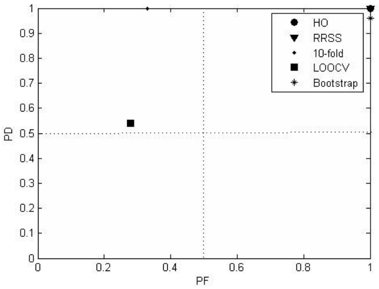


Figure 7.10: (PF, PD) pair data for the CM1_req data set in the ROC space. HO, RRSS, 10-fold, LOOCV, bootstrap are short for hold-out validation, repeated random sub-sampling, 10-fold cross-validation, leave-one-out cross-validation and non-parametric bootstrapping.

7.4.2 AUC statistic

Table 7.9 shows the empirical comparisons among the resampling methods in terms of mean AUC values. The resampling methods providing the best AUC for a particular data set is highlighted in bold face. bootstrapping results in higher AUC values for two data sets (AR1 and PC1_req) while hold-out cross-validation, 10-fold cross-validation and leave-one-out cross-validation results in the highest AUC value for one data set each.

bootstrapping AUC values suggest that it might be the most useful resampling

Table 7.9: Hold-out test set results of five resampling methods in terms of the AUC.

Data set	Resampling methods				
	Hold-out	Repeated random sub-sampling	10-fold CV	LOOCV	bootstrapping
AR6	**0.63**	0.46	0.49	0.59	0.56
AR1	0.5	0.48	0.5	0.49	**0.53**
PC1_req	0.5	0.5	0.5	0.53	**0.55**
JM1_req	0.5	0.57	**0.75**	0.53	0.49
CM1_req	0.5	0.52	0.42	**0.61**	0.5

method but we need to test for any significant differences in the AUC values. This is achieved by using the Kruskal-Wallis test, which is a non-parametric alternative to analysis of variance. It is used to test the null hypothesis H_0 that k independent samples are from identical populations. The Kruskal-Wallis test statistic h is computed as follows:

$$h = \frac{12}{n(n+1)} \sum_{i=1}^{k} \frac{r_i^2}{n_i} - 3(n+1) \tag{7.1}$$

where $n =$total number of observations; $k =$number of independent samples; $r_i =$sum of rank corresponding to n_i observations in the ith sample. If h falls in the critical region $H > \chi^2_{0.05}$ with $\nu = k-1$ degrees of freedom, null hypothesis H_0 is rejected at 0.05 level of significance [331].

The h statistic came out to be 9.52 which lies in the critical region ($h > 9.49$ for $\nu = 4$ degrees of freedom and 0.05 significance level). Therefore we can reject the null hypothesis that the samples are from identical populations. Now one can proceed with a post hoc test to determine which particular comparisons differ significantly. We used Wilcoxon rank sum test with Bonferroni correction for the pair-wise comparisons. The Bonferroni correction is a multiple-comparison correction, which is used when performing several tests simultaneously. However, performing several tests simultaneously also mean that the alpha value needs to be lowered to account for the number of comparisons being performed. One way of doing this is to use an alpha value divided by the number of comparisons. Since we compare five resampling methods, we have $\frac{5(5-1)}{2} = 10$ comparisons to make. Therefore we evaluate the comparisons using one-tenth of the 5% significance level i.e., $\alpha = 0.005$. It is observed that there is no statistically significant differences between the different resampling methods i.e., the respective p-values are greater than 0.005.

7.5 Analysis and discussion

The results of the Wilcoxon rank sum tests represent an interesting outcome. Although there are a few differences between the AUC values for the different resampling methods (Table 7.9) they, however, do not differ significantly. There can be multiple reasons for such an outcome and what we discuss here is intended to be suggestive rather than definitive:

Imbalanced data sets Data sets having a high proportion of either fault-prone or non-fault prone components would be likely to have non-significant performance outcomes no matter what resampling method is used. This is because the performance outcome of the dependent variable $y\varepsilon\{\text{fp}|\text{nfp}\}$ occurring most of the times would dominate the classification.

In our study the AR1 data set has 6.61% of records representing fault-prone while the CM1_req data set consists of 77.5% of records representing fault-prone. For JM1_req, with 45.94% of records representing fault-prone (thus representing a more balanced representation), 10-fold cross-validation is able to achieve an impressive AUC value of 0.75. This reasoning is also supported by a study by Kohavi [194] who recommends using stratification i.e., the folds are stratified so that they contain approximately the same proportions of the labels as the original data set.

Learning from imbalanced data sets is a well-known problem in machine learning. There are a number of proposed methods to resample the data in ways that diminish the effect of class imbalance. Oversampling methods involve creating data for the minority class to reach a size close to that of the larger class while undersampling methods eliminate larger class members to match the size of the smaller class. A study by Pelayo and Dick [267] show that oversampling minority class examples improved the classification accuracy using $C4.5$ decision-tree classifier. While oversampling methods might improve classification accuracy, one has to be mindful that creation of ‘synthetic’ data might lack in appeal for a practical use of a classification algorithm. Stratification *within* the data set might be a more useful alternative in this case.

Insignificant predictor variables In case of a weak relationship between the predictor variables and the dependent variable, the performance outcomes are less dependent on the resampling methods used since the potential of these resampling methods would not be utilized optimally. The data sets PC1_req, JM1_req and CM1_req contain requirement metrics which showed a weak relationship in classifying fault-proneness in a study by Jiang et al. [148]. Therefore it is more likely that the resampling methods perform non-significantly on these data sets. For data sets AR6 and AR1 containing code attributes, the relationship with fault-proneness is present but is limited.

High dimensional data set With a data set having large features, feature selection is an important task. In this study the feature selection was not performed before running the GP algorithm, primarily to exploit GP's own feature selection capability [201]. However with high feature space the efficiency and effectiveness of GP (like any other machine learning technique) can dramatically drop [272]. This potentially minimizes the impact of a particular resampling method used.

We may conclude from above reasonings that the nature of the data sets play an important role in the classification which also has an influence on the performance of the resampling methods. Secondly the selection of a non-deterministic algorithm like GP is shown to have a minimal dependency on the resampling method chosen, though this behavior is also related to the actual nature of the data sets as already discussed above.

Moreover the sample size has an impact on the resampling methods. For comparatively smaller data sets (JM1_req, CM1_req) hold-out validation is clearly not a good choice due to bias resulting from a reduced training set size, this result being in agreement with the study by Green and Ohlsson [114]. LOOCV performs better in smaller data sets due to the optimum use of the training data (0.53 and 0.61 AUC values for JM1_req and CM1_req respectively), while for larger data sets (AR6, AR1, PC1_req) it is interesting to find that there is not much difference between 10-fold cross-validation and LOOCV. Therefore 10-fold cross-validation might be more preferable to LOOCV for larger data sets. Such a choice would also guard against the potential large variance in the error estimate for LOOCV caused by a evaluating against only a single point in the test set [276].

In terms of location of (PF, PD) pairs in the ROC space, bootstrapping appears to be better placed (twice in the preferred region *A*). This indicates that bootstrapping should be considered as a resampling method for classification studies. In the context of software effort prediction studies, Kirsopp and Shepperd [186] argue that bootstrapping suffers from the disadvantage that resampling with replacement might not fit well with the practical use where there will not be multiple copies of the same project. This argument, however, holds less for software fault prediction studies where certain independent variables relate to fault-proneness rather than project-level outcome; therefore it is more realistic to have multiple records of the same component within the scope of a single project.

In statistics, there are existing studies to show that bootstrapping offers significant improvements over cross-validation [110, 333]. It is fitting that the classification studies in software engineering learn from these positive results. It is also important that the use of bootstrapping (and its variants) are evaluated more for publicly available software engineering data sets. The work of Mittas and Angelis [238] is in the right direction and more such studies are required.

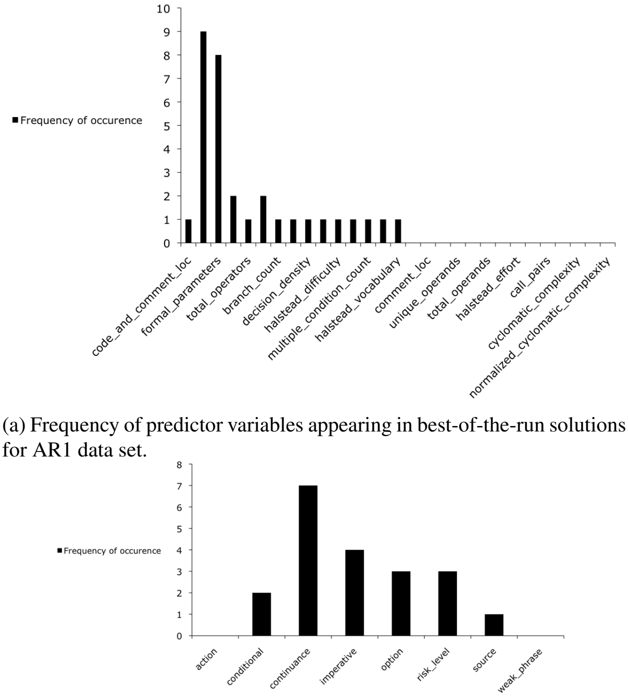
Table 7.10 shows the best individuals from the resampling methods that gave the highest AUC values for different data sets. For AR6, Table 7.10 shows the best-of-the-run individual for hold-out validation; for AR1 and PC1_req Table 7.10 shows the best-of-the-run individuals for each of the 10 bootstrap samples. For JM1_req the best-of-the-run individuals for each fold of 10-fold cross-validation are given; while for CM1_req, 10 random individuals are shown as a result of LOOCV. While the non-deterministic nature of GP is evident from Table 7.10 (producing different models for different runs of the algorithm), it is interesting to note the use of fewer predictor variables for predicting an outcome. This reinforces our earlier comment that having reduced dimensionality of data sets with fewer significant predictor variables would allow the GP algorithm to explore much richer combinations of functions and predictor variables having better fitness values. We take the case of best individuals from two data sets, AR1 and JM1_req. For AR1, an analysis of the best-of-the-run individuals reveal that only 15 of the 29 predictor variables make up the most fit solutions (they appear once or more in the solutions in Table 7.10).

Out of these 15 predictor variables there are some that appear more frequently than others, indicating their greater significance in the prediction outcome. This is shown in Figure 7.11a. For JM1_req, although we have a smaller set of predictor variables, still there exists a subset, which is more significant than its parent. This is depicted by the fact that only six out of eight variables appear in the best-of-the-run solutions (Figure 7.11b).

All what we discuss is related to using GP as a software quality classification approach. An important question to raise here is that whether or not there are differences in resampling methods when other techniques are used? For linear regression, it is shown in [50] that *k*-fold cross-validation performs better than LOOCV for model selection and evaluation; while bootstrap gave an edge in terms of model evaluation when compared with cross-validation. Using decision-trees and Naïve Bayes, Kohavi [194] showed the favorability of 10-fold cross-validation. Green and Ohlsson [114] found cross-validation and hold-out (cut-off 0.25, 0.50) as being able to better estimate the true performance of artificial neural network ensembles. While it is hard to find synergies with respect to a chosen classification technique from the above results *k*-fold cross-validation, however, scores better in general. Our study did not find 10-fold cross-validation and LOOCV to be giving significantly better results in comparison with other resampling methods, but they showed some promise in the use of the boot-

Table 7.10: Best-of-the-run results from different data sets using the resampling method with highest AUC value.

Data set	Resampling method with highest AUC	Best-of-the-run
AR6	hold-out	*sin*(*times*(total_operands,*times*(normalized_cyclomatic_complexity,code_and_comment_loc)))
AR1	bootstrapping	*times*(code_and_comment_loc,blank_loc) *sin*(*times*(blank_loc,*times*(blank_loc,*times*(blank_loc,formal_parameters)))) *times*(*times*(formal_parameters,*times*(halstead_error,halstead_error)),*times*(total_operators,*times*(halstead_error,halstead_level))) *sin*(*times*(*times*(*times*(branch_count,*times*(formal_parameters,blank_loc)),*times*(blank_loc,halstead_error)),halstead_level)) *sin*(*times*(blank_loc,*times*(cyclomatic_density,*times*(blank_loc,formal_parameters)))) *times*(*times*(formal_parameters,decision_density),*sin*(*times*(*sin*(design_complexity),halstead_difficulty))) sin(*times*(*times*(*times*(blank_loc,formal_parameters),decision_count),multiple_condition_count)) *times*(blank_loc,*times*(sin(sin(*times*(*times*(halstead_volume,halstead_volume),*times*(mylog(halstead_vocabulary),halstead_volume)))),halstead_volume)) *times*(sin(blank_loc),formal_parameters) sin(*times*(formal_parameters,blank_loc))
PC1_req	bootstrapping	*sin*(*plus*(*sin*(weak_phrase),*times*(*sin*(risk_level),*sin*(*times*(*sin*(*sin*(*sin*(conditional))),imperative))))) *plus*(*times*(conditional,*sin*(risk_level)),weak_phrase) *sin*(*plus*(*sin*(weak_phrase),*sin*(*sin*(weak_phrase)))) weak_phrase weak_phrase weak_phrase *sin*(*times*(weak_phrase,*times*(action,*sin*(weak_phrase)))) *plus*(sin(*times*(conditional,risk_level)),weak_phrase) *times*(conditional,*sin*(risk_level)) weak_phrase
JM1_req	10-fold CV	times(sin(plus(sin(plus(continuance,sin(times(option,imperative)))),sin(continuance))),sin(continuance)) *sin*(*plus*(continuance,*times*(continuance,*cos*(continuance)))) *times*(*sin*(continuance),*cos*(*mylog*(*sin*(*plus*(*sin*(imperative),imperative))))) *sin*(*times*(*sin*(continuance),*sin*(continuance))) *sin*(*times*(*times*(*sin*(continuance),risk_level),*sin*(continuance))) *sin*(*times*(*mylog*(*minus*(source,conditional)),*sin*(*minus*(risk_level,continuance)))) *sin*(*plus*(*times*(*sin*(imperative),*sin*(continuance)),*times*(*sin*(imperative),*sin*(continuance)))) *sin*(*plus*(continuance,*times*(*cos*(continuance),continuance))) *cos*(*minus*(*cos*(*cos*(option)),*cos*(*plus*(imperative,*cos*(*cos*(option)))))) *cos*(*times*(*plus*(*plus*(*plus*(conditional,*minus*(*minus*(option,conditional),*sin*(risk_level))),risk_level),*sin*(option)),option))
CM1_req	LOOCV	cos(times(continuance,weak_phrase)) *cos*(*times*(*minus*(continuance,source),option)) *cos*(*times*(*times*(risk_level,continuance),option)) *cos*(*times*(*minus*(source,action),weak_phrase)) *cos*(*times*(weak_phrase,*plus*(*times*(continuance,cos(weak_phrase)),continuance))) *cos*(*times*(*times*(*sin*(risk_level),*sin*(option)),source)) *cos*(*mylog*(*times*(*times*(continuance,source),weak_phrase))) *cos*(*times*(*times*(weak_phrase,*cos*(*cos*(weak_phrase))),*plus*(option,*minus*(source,action)))) cos(*times*(sin(sin(*times*(option,risk_level))),risk_level)) *cos*(*times*(*plus*(continuance,option),*times*(option,*mylog*(*cos*(risk_level)))))



(a) Frequency of predictor variables appearing in best-of-the-run solutions for AR1 data set.

(b) Frequency of predictor variables appearing in best-of-the-run solutions for JM1_req data set.

Figure 7.11: Frequencies of predictor variables in best-of-the-run solutions for AR1 and JM1_req data sets.

strap method. Perhaps more empirical studies involving different variants of bootstrap and different number of folds in *k*-fold cross-validation are required to reach a confident statement about the choice of resampling method, given different classification techniques and data sets. Replication studies [258] hold promise in verifying the generalizability of existing research.

What we can summarize from this study is that there are some implications on predictive studies in software engineering:

1. We need to evaluate the use of bootstrapping more for software engineering data sets. This is particularly promising for software fault prediction and software quality classification studies where multiple copies of the same records do not

pose a threat for a practical use.

2. If bootstrapping is not in contention, LOOCV is preferred for smaller data sets and 10-fold cross-validation for larger data sets.

3. For imbalanced data sets stratification might improve the performance outcome i.e., the folds are stratified so that they contain approximately the same proportions of the labels as the original data set.

4. Using automated tool support it might be possible to report results using more than one resampling methods.

5. It is important to take into account the data set properties before making a decision about a resampling method to select. Feature selection is one of the important decision criteria.

7.6 Empirical validity evaluation

There can be different threats to the validity of study results [342]. *Conclusion validity* is concerned with a statistical relationship between the treatment and the outcome with a given significance. We used Kruskal-Wallis test at 0.05 significance level with a post-hoc test where we used Wilcoxon rank sum test with 0.05 significance level with Bonferroni correction. While it is assumed that the power of a non-parametric test is less than its parametric counterpart, we were not sure of the data satisfying the assumptions of the parametric tests; therefore we resorted to the non-parametric statistical tests. The data sets used are from different domains and of different sizes; we believe that they represent a suitable heterogeneous mix. *Internal validity* is concerned with a causal relationship between the treatment and the outcome. GP is a non-deterministic algorithm and different runs of the algorithm may give different results. Therefore, we followed a standard practice, i.e., to run the GP algorithm multiple times for different folds of the different resampling methods; the exception being the LOOCV where running GP multiple times was deemed not feasible. The selection of resampling methods to compare was motivated by two factors: firstly, commonly used methods in software engineering predictive studies and, secondly, other methods which have given good results in other domains but not necessarily tried in software engineering to a large extent. The data sets used are publicly available so that the validity of the study claims can be verified through replication. *Construct validity* is concerned with the relation between theory and observation. While there are different ways to compare different resampling methods (e.g., bias, variance, mean square error, to name a few), we chose the ones described in Section 7.3.4 after taking into account the possible drawbacks with prior approaches. Moreover, although bootstrapping might have an edge with respect to the location of (PF, PD) pairs in the ROC space, one should be mindful that bootstrapping is known not to perform well for some methodologies like empirical decision-trees [194] and artificial neural network ensembles [114] for being excessively optimistic. *External validity* is concerned with generalization of the results. While other learning algorithms could have been selected our primary motive was not to compare different algorithms for classification accuracy but to compare the different resampling methods. The selection of GP was guided by its early positive results as a software classification approach (see e.g., [215]).

7.7 Conclusion

We have reported an extensive empirical comparison of five resampling methods using GP as a classifier over five public domain data sets from the PROMISE repository. We used AUC and the location of (PF, PD) pairs in the ROC space as accuracy indicators and used statistical testing procedures for contrasting different resampling methods.

Using (PF, PD) pair data across five data sets bootstrapping gave results in the preferred region of the ROC space for two data sets, indicating that bootstrapping should be considered as a resampling method of choice in predictive studies in software engineering. However, where the statistical comparison of individual resampling methods is concerned, based on AUC, there were no significant differences. We then highlight the possible reasoning of such an outcome, attributed to imbalanced data sets, insignificant predictor variables and high dimensional data sets. Hold-out validation performs less satisfactorily for comparatively smaller data sets where LOOCV performs better due to optimal use of the training data. For comparatively larger data sets 10-fold cross-validation is a better choice as compared to LOOCV.

An interesting area of future work is to investigate the outcome of different variants of bootstrapping for different software engineering data sets. Secondly resampling methods are known to have complications when applied to time-series data, something that remains relatively less explored [296, 207]. Thirdly, an interesting area is to study the impact of different settings of GP parameters [99, 64] versus the resampling methods so that one can assess how much variability in the outcome can be attributed to each factor.

The next chapter, Chapter 8, answers our second research question of the methodological investigation part. Chapter 8 aims at benchmarking feature subset selection methods for software quality classification.

Chapter 8

Genetic programming for feature subset selection: A comparative evaluation

Submitted to Journal of Systems and Software

W. Afzal & R. Torkar

8.1 Introduction

The purpose of feature subset selection (FSS) is to find a subset of the original features of a data set, such that an induction algorithm that is run on data containing only these features generates a classifier with the highest possible accuracy [195]. There are several reasons to keep the number of features in a data set as small as possible:

1. Reducing the number of features allows classification algorithms to operate faster, more effectively [124] and with greater simplicity [144].

2. Smaller number of features help reduce the curse of dimensionality[1].

[1] The requirement that the number of training data points to be an exponential function of the feature dimension.

3. Smaller number of features reduce measurement cost as less data needs to be collected [69].

4. Feature subset selection helps to achieve a better understandable model and simplifies the usage of various visualization techniques [145].

The simplest approach to feature subset selection would require examining all possible subsets of the desired number of features in the selected subset and then selecting the subset with the smallest classification error. However, this leads to a combinatorial explosion, making exhaustive search all but impractical for most of the data sets [144]. Naturally, many feature subset selection methods are search-based [57], combined with an attribute utility estimator to evaluate the relative merit of alternate subsets of attributes [124]. Despite the general acceptance that software engineering data sets often contain noisy, irrelevant, or redundant variables [11, 69], very few benchmark studies of feature subset selection methods on industrial data from software projects have been conducted. Moreover, the use of evolutionary algorithms (e.g., genetic algorithm, genetic programming) have sporadically been investigated as feature subset selection methods [243, 294, 348] but not to an extent of comparing with state of the art feature subset selection methods, using industrial data from software projects.

This chapter provides an empirical comparison of the state of the art feature subset selection methods and an evolutionary computation method (genetic programming) on five software fault prediction data sets from the PROMISE data repository [37]. Two diverse learning algorithms, C4.5 and naïve Bayes (NB) are used to test the attribute sets given by each FSS method. We are interested in investigating if the classification accuracy of C4.5 and NB significantly differ before and after the application of the FSS methods. In order to formalize the purpose of the empirical study, we set forth the following hypothesis to test:

H_0: The classification accuracy of C4.5 and NB is not significantly different before and after applying the FSS methods, i.e., $ACC_{C4.5} = ACC_{NB}$.

H_1: The classification accuracy of C4.5 and NB is significantly different before and after applying the FSS methods, i.e., $ACC_{C4.5} \neq ACC_{NB}$.

The chapter is organized as follows. The next Section describes related work. Section 8.3 briefly describes the FSS methods used in this study while

Section 8.4 describes the data sets used, the evaluation measure and the experimental setup. Section 8.5 presents the results of the empirical study and are discussed in Section 8.6. Validity evaluation is given in Section 8.7 and the chapter is concluded in Section 8.8.

8.2 Related work

Janecek et al. [145] compared information gain, wrapper and variants of principal component analysis (PCA) methods on two different data sets. The results showed that wrappers tend to produce smallest feature subsets with very competitive classification accuracy; however, they tend to be much more computationally expensive. Hall and Holmes [124] did a benchmark comparison of six feature subset selection techniques and recommended wrapper as the best method if speed is not an issue. Molina et al. [241], Guyon and Elisseeff [120], Blum and Langley [33], Dash and Liu [81] provide good surveys reviewing work in machine learning on feature subset selection.

Menzies et al. [236] used different pruning strategies for rows and columns for estimating software effort/cost. Kirsopp et al. [185, 187] and Chen et al. [69] also highlighted the usefulness of some sort of feature subset selection for software effort estimation. Catal and Diri [65] applied correlation-based feature subset selection method on class-level and method-level metrics for software fault prediction. They showed that random forests give the best results when using this feature subset selection method. Khoshgoftaar et al. [180] and Wang et al. [332] showed good results with a feature subset selection method based on the Kolmogorov-Smirnov two-sample statistical test. In another study, Khoshgoftaar et al. [181] found that the use of a stepwise regression model and a correlation-based feature subset selection did not yield improved predictions.

8.3 Feature subset selection (FSS) methods

There are two commonly known categories of FSS methods: the filter approach and the wrapper approach. In the filter approach, the feature selection takes place independently of the learning algorithm and is based only on the data characteristics. The wrapper approach, on the other hand, conducts a search for a good subset using the learning algorithm itself as part of the evaluation function [195]. Hall and Holmes [124] provides another categorization for FSS methods, namely, those methods that evaluate individual attributes and those that evaluate subset of attributes.

We have chosen to empirically evaluate a total of seven FSS methods, two that evaluate individual attributes (information gain attribute ranking and Relief), three that evaluate subsets of attributes (correlation-based feature selection, consistency-based subset evaluation, wrapper subset evaluation), one classical statistical method for dimensionality reduction (principal components analysis) and one evolutionary computational method (genetic programming). Following is a brief description of the FSS methods used in this study.

8.3.1 Information gain (IG) attribute ranking

This FSS method evaluates the ranking of each attribute by measuring the information gain with respect to the class. Information gain is the amount of decrease in the entropy of the class, and reflects the additional information about the class provided by the attribute [124]. Each attribute is ranked based on the information gain between itself and the class. More information about the method can be found in [349]. The information gain attribute ranking is used with the ranker search method that ranks attributes by their individual evaluations.

8.3.2 Relief (RLF)

Relief is an instance-based attribute ranking scheme and assigns a relevance score to each feature based on the difference between the selected instance and the two nearest instances of the same and opposite class. This process is repeated for a user specified number of instances. Relief was later extended by Kononenko [196] to handle noise and multi class data sets. More information about the method can be found in [183, 196]. The Relief method is used with the ranker search method that ranks attributes by their individual evaluations.

8.3.3 Principal component analysis (PCA)

PCA transforms the original attributes into a set of uncorrelated variables called principal components. PCA is normally done by Eigenvalue decomposition of the data covariance matrix. The role of original variables in finding the principal components is determined by loading factors. Variables with high loadings contribute more in explaining the variance. More information about the method can be found in [159]. PCA is used with the ranker search method that ranks attributes by their individual evaluations.

8.3.4 Correlation-based feature selection (CFS)

Correlation-based feature selection [122] uses the predictive ability of each feature in a subset as well as the redundancy between them to evaluate the subset. The method assigns high scores to subsets with attributes that are highly correlated with the class (high predictive ability) and having low inter-correlation with each other (low redundancy). More information about the method can be found in [122]. Correlation-based feature selection is used with the greedy stepwise forward search through the space of attribute subsets.

8.3.5 Consistency-based subset evaluation (CNS)

Consistency-based subset evaluation use class consistency to evaluate subsets of features. A search mechanism looks for the smallest subset with consistency equal to that of the full set of attributes. More information about the method can be found in [214]. Consistency-based subset evaluation is used with the Greedy stepwise forward search through the space of attribute subsets.

8.3.6 Wrapper subset evaluation (WRP)

The wrapper feature subset evaluation conducts a search for a good subset using the learning algorithm itself as part of the evaluation function. More information about the method can be found in [195]. Wrapper subset evaluation is used with the Greedy stepwise forward search through the space of attribute subsets.

8.3.7 Genetic programming (GP)

A description of GP appears in Section 3.3 (Chapter 3) of this thesis.

The best GP program (having the minimum $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$, where e_i is the actual outcome, $e_i^{'}$ is the classification result and n is the size of the data set used to train the GP models) over the 10 runs of each fold of the 10-fold cross-validation is selected. The features making up this best GP program is then designated as the features selected by the GP algorithm. The control parameters that were chosen for the GP system are shown in Table 8.1. More information about GP can be found in [29, 272].

Table 8.1: GP control parameters.

Control parameter	Value
Population size	50
Termination condition	500 generations
Function set	$\{+,-,*,/,\sin,\cos,\log,\text{sqrt}\}$
Tree initialization	Ramped half-and-half method
Probabilities of crossover, mutation, reproduction	0.8, 0.1, 0.1
Selection method	roulette-wheel

8.4 Experimental setup

We have applied the selected FSS methods to five industrial data sets from the PROMISE repository [37]. The data sets are available in ARFF (Attribute-Relation File Format), useable in the open source machine learning tool called WEKA (Waikato Environment for Knowledge Analysis) [123]. The data sets are selected based on their variance in terms of number of instances and the number of attributes. The number of instances varies from being less than 50 up to several thousands, with the number of attributes varying from being in a single digit to nearly a hundred. The characteristics of the data sets[2] are given in Table 8.2.

Table 8.2: Characteristics of the data sets used in the study.

No.	Data set	Features			No. of classes	Train size	Test size
		all	nominal	continuous			
1	jEdit	9	1	8	2	369	CV
2	AR5	30	1	29	2	36	CV
3	MC1	39	1	38	2	9466	CV
4	CM1	22	1	21	2	498	CV
5	KC1_Mod	95	1	94	2	282	CV

In order to compare the performance of different FSS methods, the attribute sets selected by each method are tested with two learning algorithms, namely C4.5 and naïve Bayes (NB). Our motivation of selecting these two algorithms is that they represent two different approaches (C4.5 being a decision-tree learner and NB being a probabilistic learner) and are considered state of the art techniques. Also one of the previous benchmark studies [124] have used the same algorithms for comparing the effectiveness of attribute selection.

Sections 6.4.4 and 6.4.2 in Chapter 6 of this thesis gives a background on naïve Bayes classifier and C4.5 respectively.

We restrict ourselves to evaluate the performance of *binary* classifiers which categorizes instances or software components as being either fault-prone (fp) or non-fault prone (nfp). We are interested in predicting whether or not a component contains any faults, rather than the total number of faults.

We have used AUC as a measure of classification performance for the different FSS methods. A description of AUC as a measure of classification performance appear in Section 7.3 (Chapter 7) of this thesis.

For all the data sets, the AUC value averaged over 10 fold cross-validation runs, was calculated for each FSS method-data set combination before and after FSS. For

[2] A detailed description of the features and the origin of these data sets can be found at their respective locations within the PROMISE data repository: jEdit, http://promisedata.org/?p=74; AR5, http://promisedata.org/?p=71; MC1, http://promisedata.org/?p=30; CM1, http://promisedata.org/?p=3; and KC1_Mod, http://promisedata.org/?p=28.

each cross-validation fold, the FSS method reduced the number of features in the data set before being passed to C4.5 and naïve Bayes classifiers.

8.5 Results

Table 8.3 shows the results for all the data sets for FSS with naïve Bayes.

Table 8.3: FSS results with naïve Bayes.

Data set	NB	IG	RLF	PCA	CFS	CNS	WRP	GP
jEdit	0.659	**0.67**	**0.67**	0.629	**0.668**	**0.67**	0.629	**0.67**
AR5	0.907	**0.933**	**0.942**	**0.938**	**0.942**	0.866	0.875	**0.915**
MC1	0.909	**0.919**	**0.92**	0.907	0.881	0.906	0.794	**0.93**
CM1	0.658	**0.718**	**0.728**	0.653	**0.691**	**0.685**	**0.738**	**0.68**
KC1_Mod	0.78	**0.851**	**0.938**	**0.854**	**0.84**	**0.86**	**0.802**	**0.87**

This table shows the AUC statistic for each FSS method and along with the AUC statistic when no feature selection is performed (the second column). The values in bold indicate if the use of the FSS method leads to an improvement of the AUC value, in comparison with when no FSS method is used. A number of FSS methods give an improved AUC value in comparison with the original AUC value without any feature selection. However, we need to test for any statistically significant differences between the different groups of AUC values. Since we have more than two samples with non-normal distributions, the Kruskal-Wallis test with significance level of 0.05 is used to test the null hypothesis that all samples are drawn from the same population. The result of the test ($p = 0.86$) suggested that it is not possible to reject the null hypothesis and, thus, there is no difference between any of the AUC values for the different FSS methods using NB *and* the AUC values of using NB as a classifier before and after applying the FSS methods. (Table 8.4 shows the number of attributes selected by each FSS method for NB.)

Wrapper, CFS, Relief and GP produce comparable AUC values with fewer number of selected features. PCA and IG, on the other hand, tend to select a much wider range of features to provide comparable classification results using NB.

Table 8.5 shows the AUC statistic for each FSS method using C4.5 along with the AUC statistic when no feature selection is used (second column). Again, the values in bold indicate that the use of FSS method leads to an improvement of the AUC value, in comparison with when no FSS is used. The result shows that multiple FSS methods do

Table 8.4: Number of features selected by each FSS method for NB. The figures in % indicate the percentage of original features retained.

Data set	Org.	IG	RLF	PCA	CFS	CNS	WRP	GP
jEdit	9	6 (66.67%)	6 (66.67%)	5 (55.55%)	5 (55.55%)	6 (66.67%)	3 (33.33%)	2 (22.22%)
AR5	30	4 (13.33%)	2 (6.67%)	7 (23.33%)	2 (6.67%)	2 (6.67%)	1 (3.33%)	6 (20%)
MC1	39	20 (51.28%)	12 (30.77%)	14 (35.90%)	4 (10.26%)	15 (38.46%)	1 (2.56%)	4 (10.26%)
CM1	22	5 (22.73%)	3 (13.64%)	4 (18.18%)	7 (31.82%)	11 (50%)	2 (9.09%)	14 (63.64%)
KC1_Mod	95	3 (3.16%)	8 (8.42%)	17 (17.89%)	8 (8.42%)	2 (2.10%)	4 (4.21%)	7 (7.37%)
Average	39	7.6 (19.49%)	6.2 (15.90%)	9.4 (24.10%)	5.2 (13.33%)	7.2 (18.46%)	2.2 (5.64%)	6.6 (16.92%)

Table 8.5: FSS results with C4.5.

Data set	NB	IG	RLF	PCA	CFS	CNS	WRP	GP
jEdit	0.594	**0.644**	**0.623**	**0.636**	**0.612**	0.592	0.636	**0.62**
AR5	0.717	**0.817**	**0.866**	**0.763**	**0.866**	**0.757**	**0.817**	**0.797**
MC1	0.791	**0.829**	**0.796**	0.708	**0.795**	0.776	0.747	**0.854**
CM1	0.558	**0.615**	**0.587**	0.506	0.542	**0.596**	0.49	**0.644**
KC1_Mod	0.599	**0.806**	**0.684**	0.555	0.553	0.589	0.579	**0.69**

improve the classification performance across all data sets. However, the result of using the Kruskal-Wallis test with $\alpha = 0.05$ ($p = 0.628$) suggested that it is not possible to reject the null hypothesis of all samples being drawn from the same population. Thus, there is no significant difference between: *a*) Any of the AUC values for the different FSS methods using C4.5. *b*) the AUC values of using C4.5 as a classifier before and after applying the FSS methods. (Table 8.6 shows the number of attributes selected by each FSS method for C4.5.)

WRP, IG, CFS and GP produce comparable average AUC values with less number of selected features. RLF, PCA and CNS tend to select a wider range of features to provide comparable classification results using C4.5.

The above results indicate that we cannot reject our earlier stated null hypothesis, H_0, and that there are no significant differences between the classification accuracy of C4.5 and NB before and after applying the FSS methods.

8.6 Discussion

When looking at the classification accuracy achieved with different FSS methods on all the data sets, it can be observed that it is not sensitive to the type of data, i.e., we had data sets with varying degree of dimensionality and number of instances, yet the FSS methods did not differ significantly when used with NB and C4.5. The degree of variation in the AUC values for different FSS methods is less and acceptable for practical use. Though there are no statistically significant differences between the AUC values for both NB and C4.5, conservatively speaking, PCA leads to fewer improvements in the original AUC values in comparison with other FSS methods. This is true even when the percentage of variance catered for by the principal components is 95%. Thus, the percentage of the total variability of the data captured in the principal components is not necessarily correlated with the resulting classification accuracy (a result similar to the study by Janecek et al. [145]).

A previous study by Catal and Diri [65] claimed to have obtained high performance

Table 8.6: Number of features selected by each FSS method for C4.5. The figures in % indicate the percentage of original features retained.

Data set	Org.	IG	RLF	PCA	CFS	CNS	WRP	GP
jEdit	9	3 (33.33%)	4 (44.44%)	5 (55.55%)	5 (55.55%)	6 (66.67%)	5 (55.55%)	2 (22.22%)
AR5	30	1 (3.33%)	2 (6.67%)	7 (23.33%)	2 (6.67%)	2 (6.67%)	1 (3.33%)	6 (20%)
MC1	39	9 (23.08%)	19 (48.72%)	14 (35.90%)	4 (10.26%)	15 (38.46%)	1 (2.56%)	4 (10.26%)
CM1	22	2 (9.09%)	9 (40.91%)	4 (18.18%)	7 (31.82%)	11 (50%)	2 (9.09%)	14 (63.64%)
KC1_Mod	95	3 (3.16%)	7 (7.37%)	17 (17.89%)	8 (8.42%)	2 (2.10%)	4 (4.21%)	7 (7.37%)
Average	39	3.6 (9.23%)	8.2 (21.02%)	9.4 (24.10%)	5.2 (13.33%)	7.2 (18.46%)	2.6 (6.67%)	6.6 (16.92%)

with CFS for software fault prediction, while Khoshgoftaar et al. [181] found that the use of CFS did not yield improved predictions. In this study, CFS along with WRP and GP consistently select fewer attributes without degrading classification performance. Hence there is a larger set of FSS methods that can potentially be used for software fault prediction studies.

In terms of evaluating an evolutionary algorithm like GP, it is worth noting that feature selection is an implicit part of GP evolution. This enables automatic or semi-automatic selection of features during model generation. GP allows almost any combination of a number of features. Evolution can freely add/remove multiple features and can reconsider previous selections as new combinations are tried [201].

We emphasize that the feature subset selection is performed automatically without human bias. This could potentially lead to a clash with human expert because the FSS results might not make sense to an expert as some unexpected features are selected as important ones or seemingly important features are left out. Either way, FSS leads to useful insights, i.e., if FSS results are acceptable to the human expert, it means that the dimensionality of the problem can effectively be reduced. If the FSS results are not acceptable, it indicates that the information which the expert wants to extract is, in principle, not present in the data [326].

8.7 Empirical validity evaluation

External validity: The data sets used in this study are collected from ongoing industry projects. The FSS methods compared in the study represent a mix of established methods as well as a not much explored method (GP). The data sets differed in their number of attributes and sizes, thus representing a variety of data sets that could be recorded in real industrial projects. *Conclusion validity:* This empirical study was performed through by use of a 10-fold cross-validation for statistically reliable results (recommended in [194]). The performance of classifiers is compared using area under the receiver operating characteristic curve (AUC) which we motivate is a standard way of evaluating classification results. *Internal validity:* We did not pre-process the data sets in any way, rather used them as is from the PROMISE repository to encourage a replication of our results. *Construct validity:* The data sets used in this study are the ones donated by the authors of fault prediction studies. Different independent variables are used to predict the faulty components, though structural measures are widely used. Since the focus of this chapter is not to examine the predictive ability of a certain type of measures, therefore, we selected datasets based on a varying degree of the number of independent variables and the number of instances (Section 8.4).

8.8 Conclusions

Feature subset selection (FSS) methods are used to keep the number of features in a data set as small as possible. Out of the various perceived advantages of using these FSS methods (Section 8.1), this study set out to evaluate whether or not the use of FSS methods have any significant affect on the classification accuracy of software fault prediction when used with two diverse learning algorithms, C4.5 and naïve Bayes.

We compare a total of seven FSS methods, representing a mix of state of the art methods and an evolutionary computation method, on five software fault prediction data sets from the PROMISE data repository. Our findings show that the use of these FSS methods do not lead to statistically significant differences in the classification accuracies (measured using AUC) for C4.5 and naïve Bayes. However, a smaller set of methods—CFS, WRP, GP—consistently select fewer attributes without degrading classification accuracy.

We recommend that any future software fault prediction study be preceded by an initial analysis of FSS methods, not missing on methods that have shown to be more consistent than their competitors.

For future work, we recommend comparing a multi-criteria GP fitness function for FSS, e.g., one that combines cost of classification with accuracy.

The next chapter, Chapter 9, investigates the possibility of effectively scheduling bug fixing tasks to developers and testers using relevant context information.

Chapter 9

Search-based resource scheduling for bug fixing tasks

Originally published in Proceedings
of the 2nd International Symposium
on Search Based Software
Engineering (SSBSE'10)

J. Xiao & W. Afzal

9.1 Introduction

Software bugs correspond to mistakes by the programmers due to an incorrect step, process, or data definition. One estimate is that a professional programmer is responsible for 5 bugs per 1000 lines of code (LoC) written on average [269]. This might not be the case with every software application but there are always a certain number of bugs in almost every software application that causes incorrect results.

Software testing is one major bug finding activity that improves software quality to a certain extent before the software application is released to the end-users. As bugs are reported, they must be triaged in a cost-effective manner, considering the resources required to fix them and their varying degrees of requirements such as severity levels. Triage of bugs in a cost-effective manner is an important decision-making task whereby competing objectives of technical, resource and budget constraints need to be balanced to provide maximum business value for the organization.

Anvik et al. [25] report that 3,426 reports were submitted to the bug database of Eclipse open source project between Jan. 1, 2005 to Apr. 30, 2005, averaging 29 reports per day. Assuming that it takes 5 minutes to triage a report, this activity costs 2 person hours per day. This indicates that we have a need to support efficient and effective bug-assignment policies that can schedule different bug fixing tasks by taking into account the available resource constraints and bug requirements. Laplante and Ahmad [205] further emphasize the value of having an efficient and effective bug assignment policy: “Bug assignment policies can affect customer satisfaction, employee behavior and morale, resource use, and, ultimately, a software product’s success”. But triaging of bugs for repairing is fraught with challenges since the number of problem variables is diverse, e.g., the severity and priority of a bug has to be balanced with resource skills and availability for finding a reasonable bug-fix schedule.

Optimal resource scheduling for bug fixing is an example of a resource constrained scheduling problem [330]. The scheduling problem in general is NP-hard, i.e., finding optimal solutions in polynomial time is hard [266, 330]. This is because the search-space becomes vast as problem size increases or more constraints are added. These properties naturally make scheduling problems a suitable problem domain for evolutionary computation approaches like genetic algorithms.

In this chapter we attempt to (at least approximately) formalize the problem of appropriately scheduling developers and testers to bug fixing activities, keeping in view the capabilities of resources and requirements of bugs. Hence, we seek an answer to the following research question:

RQ: How to schedule developers and testers to bug fixing activities taking into account both human properties (skill set, skill level and availability) and bug characteristics (severity and priority) that satisfies different value objectives by using a search-based method such as Genetic Algorithm (GA)?

The chapter is organized as follows. Section 9.2 describes some related work. Section 9.3 presents the basis of scheduling resources for bug fixing. It specifies models of bugs and human resources. Section 9.4 discusses the design of the scheduling method using a GA. Section 9.5 presents the industrial data used while results of applying the proposed method are given in Section 9.6. The GA approach is compared with hill-climbing search in Section 9.7. Section 9.8 presents a discussion while Section 9.10 presents conclusions and suggests future work.

9.2 Related work

Search techniques have been successfully used to solve different scheduling related software project management problems [132], such as software project planning [17, 24, 68, 345], software project effort prediction [186] and software fault prediction [11]. Hart et al. [133] have written a review on evolutionary scheduling; however, the application of search techniques for implementing an efficient bug repair policy is very much unexplored.

A study by Mockus et al. [239] predicted defect effort schedule based on observed new feature changes. They fitted a probability model to the observed data from eleven releases of a large real-time high availability software system and found the predicted effort to be close to reality. Cubranic and Murphy [78] applied a naïve Bayes classifier to automatically assign the bug reports to developers and achieved a 30% classification accuracy for reports entered into Eclipse's bug tracking system between Jan. 1, 2002 and Sep. 1, 2002. Zeng and Rine [353] used a self-organizing neural network approach to estimate defects fix effort. A feature map, having different clusters, was created after training the weights of the self-organizing neural networks. They computed the probability distributions of effort from the clusters and then compared them with those from the test set. For projects having similar development environments, the approach gave acceptable performance with average mean relative error (MRE) values between 7% to 23%. Canfora and Cerulo [62] used a probabilistic text similarity approach to assign change requests to developers. Song et al. [300] presented association rule mining based methods to predict defect correction effort. Using data from more than 200 projects, their approach was found to be better than partial regression trees (PART), C4.5 and naïve Bayes. Anvik et al. [25] applied support vector machine algorithm as a text categorization technique to suggest assignment of a new bug report to a small number of developers. Precision levels of 57% and 64% were obtained for the Eclipse and Firefox development projects. Recently, Weiss et al. [337] used a text similarity technique to predict bug fixing effort based on title and description of bug. Their approach beat the naive approach using the defect data from the JBoss project.

This chapter differs from related work in some important ways. Since software development is human-dependent, this work incorporates human factors such as competencies and available time-slots to schedule resources for bug fixing. This is done by using models for the bugs and human resources; moreover the use of a search-based technique such as GA is presented, which can bring a near-optimal value in scheduling by balancing multiple competing objectives.

9.3 The bug fixing process

A typical bug-tracking system such as Bugzilla [55] keeps track of a reported bug through assigned status. So the bug is marked 'new' when reported, 'assigned' when assigned to a developer for fixing, 'verified' when testing of the bug fix is done and 'resolved' when the bug is closed. This is in line with the anomaly (bug) classification process proposed by the IEEE standard classification for software anomalies (IEEE Std 1044–1993) [140], whereby the bug life-cycle is divided into four steps: recognition; investigation; action; and disposition. If we assume that a bug is valid (i.e., it is not a duplicate, not incomplete/needing more information and therefore is required to be fixed), the following events describe one instance of the above four steps in more detail (also shown in Figure 9.1):

1. A new bug is reported in the bug database which has been evaluated as a valid bug.
2. The bug is assigned to a developer for fixing.
3. The developer fixes the bug.
4. The bug is assigned to a tester for verification.
5. The bug is verified and closed or alternatively is reopened due to an incorrect fix.

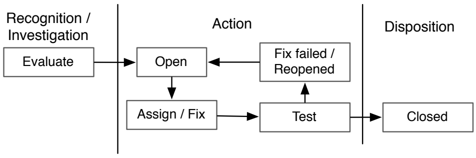


Figure 9.1: The bug fixing process.

We have restricted the scope of this chapter to schedule resources for a single round of these five events. So if a bug-fix from a developer fails at testing and is re-opened, a second round of events need to be taken; however, this is not dealt with in this chapter since it is not known in advance how many of these round of activities would be required.

It is clear from the different bug life-cycle events that given a number of bugs reported in the bug data base, there are two resource consuming activities taking place:

development activity for fixing bugs and testing activity for verifying these bug-fixes. The criticality and resource demands for various bugs require resources with desired competencies and skills; moreover this has to be balanced with availability/workload of resources for getting the job done. Due to that common constraint of limited resources to play with, and engagement of resources in multiple projects concurrently, the bug fixing events are in a competition for finding resources that have the availability and competence to fix/verify reported bugs. To schedule the capable and available resources, to balance the competing objectives and to bring near-optimal value by using scheduling, we need a degree of formalism to describe the reported bugs and required human resources. This is done by describing a bug and a human resource model.

9.3.1 The bug model

This chapter only focuses on the bugs found during system testing. This is however more of a constraint rather than a rule and our proposed methodology should be equally applicable to bugs found at other testing levels.

We define a bug data repository, *BR*, as a collection of reported bugs, $BR = \{B_1, B_2, \ldots, B_n\}$, where each bug B_i in this repository has the following attributes:

1. *Bug ID:* A unique identification of the bug.

2. *Bug description:* A short description of the bug.

3. *Bug severity:* The perceived impact of the bug, having possible values of High, Medium and Low.

4. *Bug priority:* A classification indicating the order in which the bugs are addressed, having possible values of High, Medium and Low.

5. *Required skills:* The skills required for bug fixing, which are used to select the candidate resources. Each required skill is described as a triplet (SKT, SKN, SKL), where

    (a) SKT: The type of required skills, which in our context are two, namely development skills and testing skills.

    (b) SKN: The name of a specific required skill, e.g., programming language skills for the skill type: development and test design skills for the skill type: testing.

    (c) SKL: The minimum required competency in a particular skill for fixing/ verifying the bug, having possible values of Low, Medium and High.

It is to be noted that the definition of skills required to fix/verify a bug would be different across software companies; however the skill structure defined above is flexible to incorporate different skill types.

6. *Estimated effort for fixing the bug:* The estimated required effort, in number of person-hours, to be invested in development and testing of the bug-fix. Note that this estimated required effort is for one round of events, as described in Section 9.3.
7. *Assigned time:* The date when the bug fixing activity can be started.
8. *Deadline for bug fixing:* The date by which the bug has to be fixed and verified. This attribute is used as a constraint for scheduling.
9. *Actual bug fixing time:* The actual date when the bug fixing is finished. This includes both the actual development and the actual testing time taken by a bug for resolution. This is given by the scheduling results.
10. *Coefficient of schedule benefit (CSB):* If the actual bug fixing time is before the deadline for bug fixing, there is an incurred benefit given by CSB which is described by the benefit for each day before the deadline.
11. *Coefficient of schedule penalty (CSP):* If the actual bug fixing time is later than the deadline, it is expected that some other work activity would get affected. Thus there is a penalty, CSP, involved in this case for each day used up later than the deadline.

    The values for the coefficients *CSB* and *CSP* are configurable parameters chosen accordingly by the stakeholders. For instance, the penalty in missing a deadline might have higher impact than the benefit in fixing the bug before deadline; so *CSP* might get a higher stakeholder value than *CSB*.

From the above attributes we can determine the value of each bug based on the following value function:

$$\begin{aligned} Value(B_i) = f(&B_i.priority, B_i.severity, B_i.deadline,\\ &B_i.actual\_fixing\_time, B_i.CSB, B_i.CSP) \end{aligned}$$

Among these parameters, actual bug fixing time is decided by the scheduling results and the others are determined by the value objectives of stakeholders before the scheduling. The summation of the values of all the bugs provides us with the overall value of the bug fixing process:

$$Value(BS) = \sum_{i}^{n} Value(B_i)$$

9.3.2 The human resource model

The human resource model captures the competencies and availability of development and testing personnel to undertake bug fixing/verification. We make use of the human resource model proposed by Xiao et al. [346] to describe different attributes of human resources in the bug fixing process. A human resource, HR, is defined in terms of four attributes:

1. HR ID: A unique identification of the human resource.
2. SKLS: The set of skills possessed by a human resource, $SKLS = \{skl_1, skl_2, \ldots, skl_n\}$. Each skl_i $(1 \leq i \leq n)$ is defined by a triplet as $skl_i = (SKT, SKN, SKL)$. The elements in this triplet are the skill type (SKT), skill name (SKN) and skill level (SKL). For example, an experienced testing resource (SKT) might be highly competent (SKL) in a certain test design technique (SKN).
3. EXPD: The work experience figure, in number of years, for the human resource. This figure can give an indication of the skill level of a particular resource.
4. STMW: The time and the workload that can be scheduled for a human resource. STMW consists of all free time periods and the workload per day in each of these time periods:

   $$STMW = \{([T_{s1}, T_{e1}], W_1), ([T_{s2}, T_{e2}], W_2), \\ \ldots, ([T_{sk}, T_{ek}], W_k), \}$$

   where T_{si} and T_{ei} represent the start and end date of the i^{th} free time period respectively, W_i represent the workable hours per day that can be scheduled in the i^{th} free time period. The unit for W_i is person hour. For example, $([25 - Mar - 2010, 07 - Apr - 2010], 6)$ indicates that the resource is available between $25 - Mar - 2010$ and $07 - Apr - 2010$ for 6 hours per day, excluding the weekends.

If a human resource has all the skills required for fixing a bug, then depending upon the available time periods, this human resource can be scheduled for the bug fixing task.

Thus, according to the human resource descriptions and skill requirements of bugs, the capable resources for each bug fixing event/activity can be scheduled.

However, the organizations might lack the required competencies for fixing certain bugs within the stipulated deadline, e.g., if the verification of a bug-fix requires a high skill level of domain knowledge on part of the testing resource but the one available has medium or low skill levels. In such a scenario, we setup certain rules aimed at relaxing the skill requirements in order to provide additional capable candidates for bug fixing. Thus the organization can take a risk of lowering the skill requirements in an attempt to close a bug on deadline. Following are the rules to relax the skill requirements and provide additional candidate capable resources for bug fixing if:

- the skill level gap between what is required and what is available is less than a specific number such as '1'. This number indicates the scale of gap, so e.g., the gap is '1' if there is a requirement of high level of a certain skill but only a medium one is available.

- the number of skills possessed by resources, having levels lower than the requirement, is less than a given value, e.g., '3', that is, at most a resource is lacking in '3' skill levels than what is the requirement.

9.4 Scheduling with a genetic algorithm (GA)

Scheduling resources for bug fixing activities represent a problem with different competing constraints and even with a moderate number of bugs, the search space can become vast as the number of combinations grows. To deal with the complexity of such a combinatorial optimization problem we apply a genetic algorithm (GA).

GA is an evolutionary algorithm that uses simulated evolution as a search strategy to evolve potential solutions and uses operators inspired by genetics and natural selection [136]. A GA encodes the candidate solutions to the search problem as finite length strings called chromosomes. The chromosomes are made up of components called genes while the values of these genes are called alleles. A fitness measure discriminates good candidate solutions from bad ones and guides the search towards feasible areas in the search space. A genetic algorithm maintains a population of solutions, which is iteratively recombined and mutated to evolve successive generations of candidate solutions.

9.4.1 Encoding and decoding of chromosome

Before encoding the scheduling problem as a GA chromosome, the following assumptions are made:

- Only one development resource and one testing resource can be allocated to each bug.
- One developer can only fix one bug at a time. Similarly one tester can verify one bug-fix at a time.

We use a binary representation of integers to encode the bug fixing problem as a chromosome, an approach similar to the one in [345]. We establish a set of resource genes and a set of priority genes. For each bug B_i in bug repository $BR = \{B_1, B_2, \ldots, B_n\}$, the fixing of bug B_i includes two activities, development (DEV) and testing (TST). For each of these two activities, there are number of r_i capable (or additional candidate capable) resources that can be allocated to it. We encode these capable resources as a set of binary genes, where the size of the set is the smallest integer greater than or equal to $log_2 r_{i,j}$ where i represents bug i and j represents activity type (DEV or TST). The binary values of these genes are used to represent the decimal number that identifies a scheduled resource as shown on the left part of Figure 9.2.

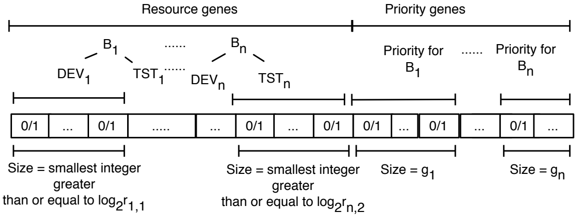


Figure 9.2: The bug fixing chromosome structure.

When two or more activities described by resource genes contend for the same resource, which activity can first acquire resources should be determined. Thus, a group of genes named as priority genes are set, describing the activity priority (shown on the right part of Figure 9.2, where g is the priority gene size for each bug). The activity with higher priority is assigned to the resource first while if two activities have the same priority, the one placed to the left in the chromosome is assigned to the resource first.

Decoding is the reverse process of encoding. First, resource genes of each activity are decoded to a real number, giving us scheduled resource for the activity. Second, the priority genes for each activity are decoded to a real number, giving us the priority of each activity. Third, the start time and end time for each activity is calculated. This calculation satisfies the following constraints:

1. If two activities require the same resource, the one with higher priority will be scheduled first.
2. The availability constraints of the human resources should be satisfied.

9.4.2 Multi-objective fitness evaluation of candidate solutions

Each scheduling result decoded by a chromosome is evaluated by means of a fitness function. The fitness function is designed to keep in view the scheduling objectives. Two generic and two specific objectives are taken in to consideration for scheduling. The generic objectives are:

- Objective 1: Bugs with higher priority and severity bring higher value on fixing.
- Objective 2: From a scheduling perspective, the maximum total value of fixing all the bugs should be obtained.

Besides these generic objectives, two specific objectives are used, each bringing a different value return for getting a bug fixed. One specific objective is the strict deadline objective:

- Objective 3: The deadline for each bug is strict. If a bug cannot be fixed before a deadline, the value for fixing this bug is minimum, i.e., 0. If it can be fixed before deadline, the value for fixing it is computed by its priority, severity and preference weight.

By using these objectives the value of a bug B is described as follows:

$$Value(B) = (\alpha * priority + \beta * severity) * HasFinished(B)$$

where α and β are the preference weights for priority and severity respectively; $HasFinished(B)$ is an operator:

$$HasFinished(B) = \begin{cases} 1 & \text{B is fixed before deadline} \\ 0 & \text{B cannot be fixed before deadline} \end{cases}$$

The other specific objective is the relaxed deadline objective:

- Objective 4: If bug fixing is finished before the deadline, there is an incurred benefit. If bug fixing is finished later than deadline, a penalty is applied.

By using objectives 1, 2 and 4, the value of a bug *B* is described as follows:

$$Value(B) = (\alpha * priority + \beta * severity) * ScheduleValue(B)$$

where α and β are the preference weights of priority and severity respectively, while $ScheduleValue(B)$ is computed as follows:

$$ScheduleValue(B) = \begin{cases} (B.Deadline - B.FixedTime) * B.CSB \\ \qquad\qquad for Deadline \geq FixedTime \\ (B.Deadline - B.FixedTime) * B.CSP \\ \qquad\qquad for Deadline < FixedTime \end{cases}$$

where *CSB* and *CSP* are configurable parameters, set by the user. The strength of these coefficients indicate the impact of benefit or otherwise on the bug fixing process so e.g., if the impact of missing a deadline is more, the corresponding coefficient is set to a higher value.

No matter whether the deadline of a bug is strict or not, the value function for the bug fixing process is:

$$Value(BS) = \sum_{i}^{n} Value(B_i)$$

This value function is used as a fitness function during the GA evolution process.

9.5 Industrial case study

Our proposed methodology for scheduling resources for bug fixing activities is evaluated using data from large Enterprise Resource Planning (ERP) software developed by a global provider of geo-technology and information technology services. The company consists of over 600 skilled professionals and have successfully been certified as CMMi level 3 compliant. The ERP project has completed several releases while the data used in this chapter comes from a batch of bugs reported by the testing team for an upcoming release. This upcoming release incorporates customized functionality for one of their telecom clients. The project team working on the upcoming release have to schedule appropriate resources to cut-down the back-log of reported bugs from the testing team. The project leader plans for fixing every bug by a set deadline (keeping in view the release date for the customer) and estimates the required effort using expert judgement. The project leader is responsible for triaging the bugs to resources having the required skills (along with required skill levels) and available times. The skill set and associated levels for every resource in the project is maintained by the human resource department and the project leader also has own qualitative assessments regarding the skill levels of resources under him. The empty time slots for every resource are available through a centralized calendar application.

Therefore, having bugs with different priority, severity, time constraints, resource constraints and having resources with varying skill sets with associated skill levels and available time slots, an automated mechanism to triage bugs with maximum possible value for the organization is required.

9.5.1 Description of bugs and human resources

We evaluated our approach on a set of 25 bugs logged in the bug repository during system testing. The ID, description, severity, priority, assigned time, deadline and estimated effort are shown in Table 9.1. The bug descriptions have been modified to protect privacy. As discussed in Section 9.3, there are two resource consuming activities taking place during the bug fixing process: development (DEV) and testing (TST); each of these activities require relevant skill sets. Although there can be different ways of classifying skills required for both development and testing, we use more general skill requirements that could easily be mapped to more specific skill-set at our subject company. The skill requirements for each bug are described in Table 9.2 where H: High, M: Medium and L: Low. Human resource attributes for available development and testing personnel (as discussed in Section 9.3.2) are shown in Table 9.3.

9.6 The scheduling results

We applied the GA proposed in Section 9.4 to schedule capable resources for bug fixing activities, based on the bug model and human resource model data given in Section 9.3. The GA used the following parameters: population size, 100; total number of generations, 500; cross-over rate, 0.8; mutation rate, 0.01; selection method, ratio. These parameters were obtained after some experimentation; however, in the future we need a more systematic mechanism of tuning them, perhaps using an automated way.

We assume that delaying the bug fixing after the deadline has greater impact than fixing it before, therefore, *CSB* is set as 10 and *CSP* as 30 for every bug, i.e., one day delay in bug fixing has three times effect on the value than completing the bug fixing one day before. Based on the configuration of coefficients and weights in balancing objectives (Section 9.4.2) and resources (Section 9.5.1), different scenarios suggest strategies

Table 9.1: Bug descriptions.

Bug ID	Bug Description	Severity (H:Hi, M: Med, L:Low)	Priority (H:Hi, M: Med, L:Low)	Assigned time	Deadline	Estimated effort
1	Reservations removed.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
2	Built-in redundancy is lost.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 4 days; 32 Hours
3	Replication too slow.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
4	Scheduled periodic account management job not working.	H	H	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
5	The scheduled job cannot perform evaluation.	H	H	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
6	Modifying existing schedule not allowed.	H	H	01-Apr-10	19-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
7	History of customer profile not loaded.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 4 days; 32 Hours
8	Too low processing performance.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
9	Req002 not fulfilled.	H	H	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
10	Volume input/output not working.	M	H	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
11	CustomerHandler crashed.	M	M	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
12	Fallback fails in step 2 of use case 1.	H	M	25-Mar-10	12-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
13	User data not updated in customerHandler memory.	H	M	01-Apr-10	19-Apr-10	DEV: 4 days, TST: 2 days; 48 Hours
14	Failed to generate customer request.	M	M	25-Mar-10	12-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
15	Configuration file corrupted.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
16	The configuration log is missing latest settings.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
17	Too low performance for handling batch requests.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
18	Page not displayed on server authentication.	L	M	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
19	Notification email not send.	L	M	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
20	Database replication error.	M	M	01-Apr-10	19-Apr-10	DEV: 3 days, TST: 1 day; 32 Hours
21	Incorrect error code.	L	L	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
22	Incorrect salary shown.	L	L	25-Mar-10	12-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
23	Report taking too long to generate.	L	L	01-Apr-10	19-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
24	Message update required.	L	L	01-Apr-10	19-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours
25	Usage profile not updated.	L	L	01-Apr-10	19-Apr-10	DEV: 1.5 days, TST: 0.5 day; 16 Hours

Table 9.2: Skill requirements of each bug.

	Development skills						Testing skills						
Bug ID	Analytical	Programming language	Debugging	Refactoring	Use of IDE	Configuration management	Use of libraries and frameworks	Test planning (TP)	Test design (TD)	Test execution (TE)	Test review (TR)	use of bug tracking tool	Domain knowledge (DK)
1	H	M	H	M	M	M	M	H	H	H	M	M	H
2	H	M	H	M	M	M	M	H	H	H	M	M	H
3	H	M	H	M	M	M	M	H	H	H	M	M	H
4	H	H	H	H	M	M	M	H	H	H	M	M	H
5	H	H	H	H	M	M	M	H	H	H	M	M	H
6	H	H	H	H	M	M	M	H	H	H	M	M	H
7	H	M	H	M	M	M	M	H	H	H	M	M	H
8	H	M	H	M	M	M	M	H	H	H	M	M	H
9	H	H	H	H	M	M	M	H	H	H	M	M	H
10	H	M	H	M	M	M	M	H	H	H	M	M	H
11	H	M	H	M	M	M	M	H	H	H	M	M	H
12	H	H	H	H	M	M	M	H	H	H	M	M	H
13	H	H	H	H	M	M	M	H	H	H	M	M	H
14	H	M	H	M	M	M	M	H	H	H	M	M	H
15	H	M	H	M	M	M	M	H	H	H	M	M	H
16	H	M	H	M	M	M	M	H	H	H	M	M	H
17	H	M	H	M	M	M	M	H	H	H	M	M	H
18	M	M	M	L	L	L	L	M	M	M	L	L	M
19	M	M	M	L	L	L	L	M	M	M	L	L	M
20	H	M	H	M	M	M	M	H	H	H	M	M	H
21	M	M	M	L	L	L	L	M	M	M	L	L	M
22	M	M	M	L	L	L	L	M	M	M	L	L	M
23	M	M	M	L	L	L	L	M	M	M	L	L	M
24	M	M	M	L	L	L	L	M	M	M	L	L	M
25	M	M	M	L	L	L	L	M	M	M	L	L	M

Table 9.3: Human resource descriptions.

HR ID	(SKT, SKN, SKL)	EXPD (Years)	STMW
HR1	(Developer, Analytical, H)	6	([25-Mar-2010, 07-Apr-2010], 6) ([12-Apr-2010, 15-Apr-2010], 6)
	(Developer, Programming Lang., H)		
	(Developer, Debugging, H)		
	(Developer, Refactoring , H)		
	(Developer, IDE, M)		
	(Developer, CM, M)		
	(Developer, Lib. and Frameworks, M)		
HR2	(Developer, Analytical, H)	6	([20-Mar-2010, 04-Apr-2010], 8) ([12-Apr-2010, 14-Apr-2010], 8)
	(Developer, Programming Lang., H)		
	(Developer, Debugging, H)		
	(Developer, Refactoring , H)		
	(Developer, IDE, M)		
	(Developer, CM, M)		
	(Developer, Lib. and Frameworks, M)		
HR3	(Developer, Analytical, M)	2	([23-Mar-2010, 12-Apr-2010], 8) ([15-Apr-2010, 22-Apr-2010], 8)
	(Developer, Programming Lang., M)		
	(Developer, Debugging, M)		
	(Developer, Refactoring , M)		
	(Developer, IDE, M)		
	(Developer, CM, L)		
	(Developer, Lib. and Frameworks, L)		
HR4	(Developer, Analytical, M)	2	([23-Mar-2010, 05-Apr-2010], 6) ([12-Apr-2010, 17-Apr-2010], 6)
	(Developer, Programming Lang., M)		
	(Developer, Debugging, M)		
	(Developer, Refactoring , M)		
	(Developer, IDE, M)		
	(Developer, CM, L)		
	(Developer, Lib. and Frameworks, L)		
HR5	(Tester, TP, H)	4	([25-Mar-2010, 08-Apr-2010], 4) ([12-Apr-2010, 14-Apr-2010], 4)
	(Tester, TD, H)		
	(Tester, TE, H)		
	(Tester, TR, M)		
	(Tester, Bug Tracking Tool, M)		
	(Tester, DK, M)		
HR6	(Tester, TP, M)	2	([25-Mar-2010, 06-Apr-2010], 3) ([09-Apr-2010, 16-Apr-2010], 3)
	(Tester, TD, M)		
	(Tester, TE, M)		
	(Tester, TR, L)		
	(Tester, Bug Tracking Tool, L)		
	(Tester, DK, M)		

for managing resources for the bug fixing tasks. We then discuss the scheduling results out of these scenarios.

9.6.1 Scenario 1: Priority preference weight, $\alpha = 20$; Severity preference weight, $\beta = 5$

With priority weight, α, set to 20 and severity preference weight, β, set to 5, we first use objectives 1, 2 and 3 from Section 9.4.2. That is, we use the strict deadline as an objective and find that, using data from Section 9.5.1, only 11 out of 25 bugs can be scheduled for fixing. These bugs are listed in Table 9.4 and the corresponding Gantt chart plan for fixing these bugs is shown in Figure 9.3. Gantt chart is an easy way to illustrate a project schedule and provides an intuitive interface for the project leader to monitor scheduling elements. As is clear, using a strict deadline objective, only a limited number of bugs can be fixed.

Table 9.4: Bugs that can be fixed under a strict deadline (α=20, β=5).

Bug ID	Value	DEV	TST
2	70	HR2: ([25-Mar-2010, 26-Mar-2010], 8) ([29-Mar-2010, 29-Mar-2010], 8)	HR5: ([30-Mar-2010, 31-Mar-2010], 4)
3	70	HR2: ([30-Mar-2010, 01-Apr-2010], 8)	HR5: ([06-Apr-2010, 07-Apr-2010], 4)
9	75	HR1: ([29-Mar-2010, 01-Apr-2010], 6)	HR5: ([02-Apr-2010, 02-Apr-2010], 4) ([05-Apr-2010, 05-Apr-2010], 4)
10	75	HR1: ([02-Apr-2010, 02-Apr-2010], 6) ([05-Apr-2010, 07-Apr-2010], 6)	HR5: ([08-Apr-2010, 08-Apr-2010], 4) ([12-Apr-2010, 12-Apr-2010], 4)
18	45	HR4: ([25-Mar-2010, 26-Mar-2010], 6)	HR5: ([29-Mar-2010, 29-Mar-2010], 4)
19	45	HR3: ([25-Mar-2010, 26-Mar-2010], 8)	HR6: ([29-Mar-2010, 30-Mar-2010], 3)
21	25	HR1: ([25-Mar-2010, 26-Mar-2010], 6)	HR6: ([31-Mar-2010, 01-Apr-2010], 3)
22	25	HR3: ([29-Mar-2010, 30-Mar-2010], 8)	HR5: ([01-Apr-2010, 01-Apr-2010], 4)
23	25	HR3: ([01-Apr-2010, 02-Apr-2010], 8)	HR6: ([05-Apr-2010, 06-Apr-2010], 3)
24	25	HR3: ([05-Apr-2010, 06-Apr-2010], 8)	HR5: ([13-Apr-2010, 13-Apr-2010], 4)
25	25	HR3: ([07-Apr-2010, 08-Apr-2010], 8)	HR6: ([09-Apr-2010, 09-Apr-2010], 3) ([12-Apr-2010, 12-Apr-2010], 3)

We now use the relaxed deadline objective to schedule more bugs by relaxing the deadline constraint. We assume that all the resources are available after 20-Apr-2010 and each workday comprises of 8 working hours. Using objectives 1, 2 and 4 from Section 9.4.2, the simulation results appear in Table 9.5. The data in Table 9.5 indicates that relaxing the deadline enables all the bugs to be scheduled for fixing but many of them are delayed as indicated by negative integers in the third and sixth columns of Table 9.5. The corresponding Gantt chart plan is shown in Figure 9.4 and could help in showing the project leader that negotiating a relaxation in deadline would help fix all the bugs.

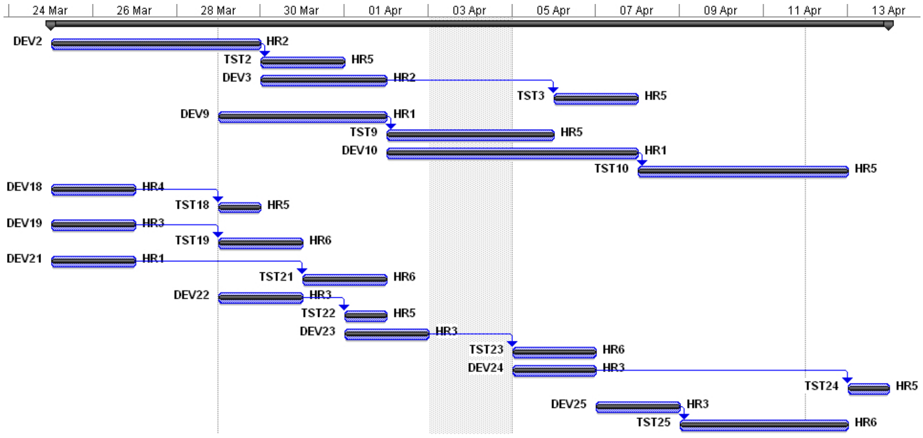


Figure 9.3: Strict deadline bug fixing plan.

Table 9.5: Bugs that can be fixed under a relaxed deadline (α=20, β=5).

Bug ID	Value	Precedent days compared to the deadline	Bug ID	Value	Precedent days compared to the deadline
1	-18900.0	-9	14	-37500.0	-25
2	-23100.0	-11	15	-24000.0	-16
3	4900.0	7	16	-33000.0	-22
4	-4500.0	-2	17	-42000.0	-28
5	-40500.0	-18	18	4950.0	11
6	-33750.0	-15	19	4500.0	10
7	-21000.0	-10	20	-25500.0	-17
8	6300.0	9	21	2000.0	8
9	3750.0	5	22	1500.0	6
10	-16800.0	-8	23	1750.0	7
11	-31500.0	-15	24	1250.0	5
12	-52800.0	-32	25	2250.0	9
13	-51150.0	-31			

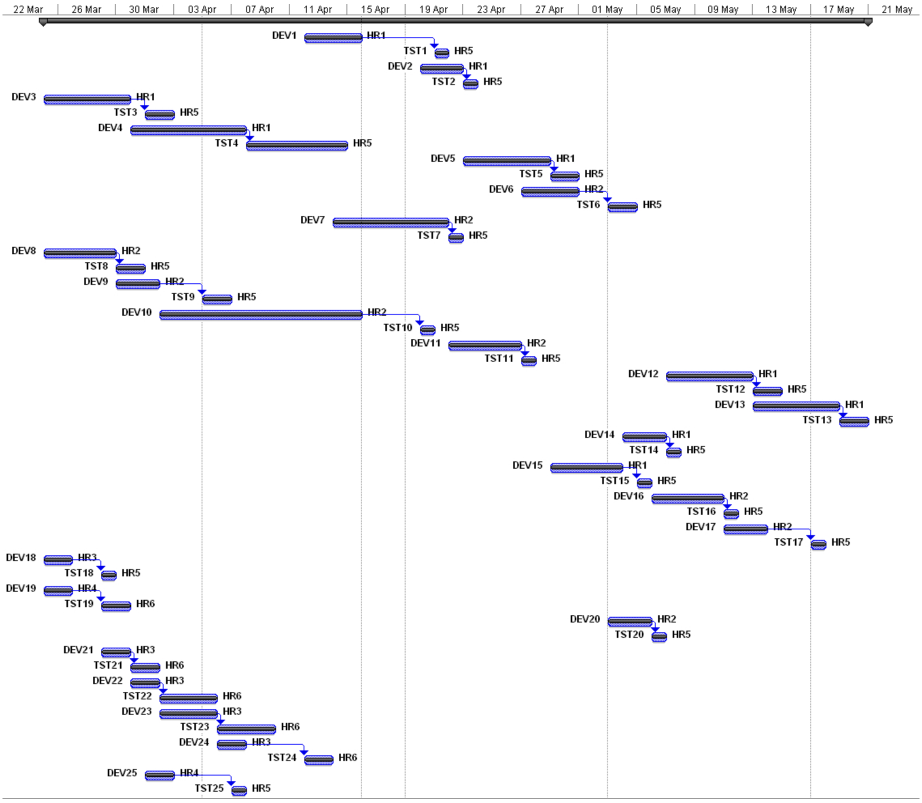


Figure 9.4: Relax deadline bug fixing plan.

9.6.2 Scenario 2: Priority preference weight, $\alpha = 5$; Severity preference weight, $\beta = 20$

In this scenario, we change the priority and severity preference weights as $\alpha = 5$ and $\beta = 20$ respectively; that is to say that we now consider severity as more important than the priority of a bug. Using the strict deadline as an objective, the simulation results indicate that there are still 11 out of 25 bugs that can be scheduled before deadline. Although the total number of bugs that could be fixed before deadline remains the same for both the scenarios, a comparison of two schedules indicate that the two scheduling plans differ at bug IDs 3, 5, 10 and 16. This is shown in Table 9.6.

Table 9.6: Comparison of bug fixing schedules under different priority and severity preference weights.

Bug ID	Priority	Severity	Value preference weight	
			α = 20; β = 5	α = 5; β = 20
2	3	2	70	55
3	3	2	70	0
5	3	3	0	75
9	3	3	75	75
10	3	2	75	0
16	2	2	0	50
18	2	1	45	30
19	2	1	45	30
21	1	1	25	25
22	1	1	25	25
23	1	1	25	25
24	1	1	25	25
25	1	1	25	25

The data in Table 9.6 show that increasing the severity preference weight β (Scenario 2) has resulted in scheduling bug ID 5 with highest priority but at the cost of not fixing bug IDs 3 and 10 from Scenario 1. Since bug IDs 3 and 10 cannot be fixed, the available resources are enough to fix bug ID 16. Scenarios 1 and 2 indicate that by plugging different combinations of priority and severity preference weights, the project leader can balance the importance of fixing certain bugs at the cost of others (provided that the deadline is strict). This, in our view, suggests valuable strategies for resource scheduling.

9.6.3 Scenario 3: Priority preference weight, $\alpha = 20$; Severity preference weight, $\beta = 5$; Simulating virtual resources

In the previous two scenarios we see that, using a strict deadline, the resources are not enough to fix all the bugs on or before the deadline. We also saw that one way to provide more candidate resources is to relax the deadline. The other way to achieve the deadline is, of course, addition of more resources. Therefore, we add virtual resources for fixing bugs in this scenario. Initially we have 6 resources and are able to schedule 11 out of 25 bugs for fixing. Increasing the resources to two more by adding one development resource and one testing resource, with high skill levels in all skills, enables scheduling over 15 bugs for fixing. Similarly increasing the resources to 10 by adding two highly-skilled development resources and two highly-skilled testing resources allow scheduling more than 20 bugs before deadline (Figure 9.5). This scenario gives another option to the project leader for viewing the scheduling outcome if more resources were available than initially assigned. Therefore, the simulation of virtual resources can provide schedules under varying circumstances, keeping in view the available resource pool.

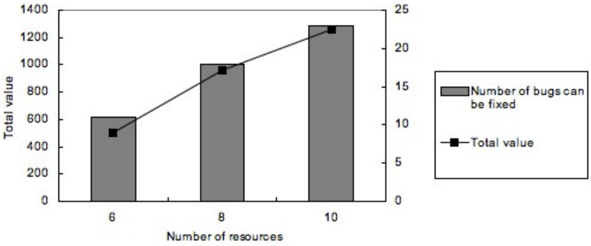


Figure 9.5: Effects of increasing number of resources.

9.7 Comparison with hill-climbing search

Hill-climbing (HC) is a basic local search algorithm and, likewise GA, is used to compute the value obtained in scheduling resources for bug fixing tasks. We have used the three scenarios discussed in Section 9.6 to compute the total bug value by using HC and have compared it with GA. The results are shown in Figure 9.6. The figure shows that if the number of bugs is small (i.e., 1 to 3), GA and HC obtain the same optimal

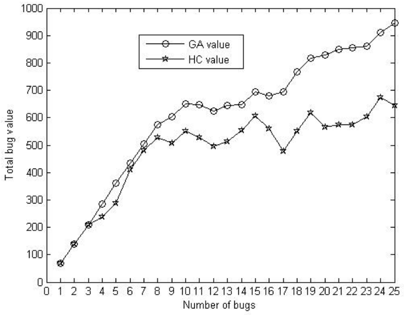


Figure 9.6: A comparison of total bug value obtained by using GA and HC.

value. But when the scale of problem increases with an increase in number of bugs, GA gives better results than HC. In order to test whether any significant differences exist between the bug values from two algorithms, we used Wilcoxon rank sum test. The p-value of 0.004 confirmed that the bug values from HC and GA do not have equal medians at $\alpha = 0.05$. Thus, bug values from GA are significantly different and better than those of HC.

9.8 Discussion

Considering that we have a need to support efficient and effective bug assignment policies, this chapter has provided early results as to how a GA can help strike a balance between competing constraints to achieve near-optimal value for the organization. Due to the dynamic nature of the bug fixing schedules, different scenarios are possible and these changing scenarios have to be modeled effectively for near-optimal solutions. The multi-objective fitness function proposed in this chapter attempts to model the uncertainty in the scheduling problem. The scenarios discussed here provide a way to schedule resources under different circumstances, e.g., having a strict/flexible deadline, having assigned different weights to severity and priority and last but not least, the ability to foresee the resource requirements by adding virtual resources to meet the deadline. GA is able to effectively suggest different strategies to tackle the bug fixing process and is found to be more effective than hill-climbing. Consequently the project leaders can use these results to support their resource scheduling decisions. We are also aware of certain limitations of our study. First we have some assumptions that might get violated, e.g., it is common that the bug-fix is verified not to be correct by the testing team and a second round of bug fixing activities is undertaken. If this is the case, then the different elements of the bug model would require new data for the second round of activities. We, however, limit ourselves to only one round of bug fixing activities.

Second, there are some rules that are followed for the relaxed deadline objective. While these rules would differ with respect to the expectations a project leader has on her team members, we followed some intuitive ones. Any change in these rules is, however, possible.

Third, there is a possibility that a resource works concurrently on more than one assignment; however, we only consider the empty time slots that a particular resource has for dealing with one bug at a time. If such a division of workload is not possible then it is expected that the human resource model needs to incorporate this change.

Fourth, a company might face the difficulty to quantify the skills and the associated levels. As our subject company is on the path of CMMi Level 4, such quantification is seen as a continuing improvement opportunity for the workforce. The human resource model presented here uses a simple classification of skills, which can be changed to suit specific needs. There is a possibility that the human resource model we propose has ignored relevant human performance factors. An important point to make here is that the company using such an approach needs to continuously update the skill database of its resources since it is common for the resources to educate themselves and learn as part of the project experience.

9.9 Empirical validity evaluation

Construct validity threats refer to the extent the experiment setting actually reflects the construct under study [342]. These threats might arise due to the assumptions we made in the study and the way we modeled the problem. However, a search-based technique such as GA is independent of the way the problem is modeled; it is the fitness function that contains the crucial information and needs to be adapted for a more complex model. The assumptions in this chapter made sense for the type of case study discussed; however, as mentioned in Section 9.8, the bug and human resource model might change if a different process of bug fixing is followed. *Internal validity* threats refer to any sources of bias that might have affected the results. Since GA is a stochastic algorithm, different runs produce different solutions. The different GA parameters were obtained after careful experimentation and taking into account that further changing the parameter values do not have significant impact on the results. Moreover, the

GA was run multiple times (30) to, partly, overcome the stochastic nature of GA. *External validity* threats are concerned with generalization. The results obtained in this chapter as such should be applicable to the situations where our assumptions are held. Otherwise, the bug and the human resource model needs to be adapted accordingly.

9.10 Conclusions and future work

We have presented a search-based resource allocation method for bug fixing tasks. We proposed models for the bug fixing process, the human resources and the capability matching method between bug fixing activities and human resources. On the basis of these models and our proposed method, the resources were allocated for bug fixing activities using a GA. Depending on differing objectives, three scenarios were discussed using an industrial data set and the results showed that GA was able to give schedules having balanced different objectives and entailing maximum value for the organization. Comparison with hill-climbing showed that GA gave statistically better results in terms of maximizing the value objective.

Based on this chapter, some interesting future work can be undertaken:

- Combining the bug fixing process with other resource consuming activities that might happen concurrently, e.g., testing of newly implemented requirements might take place in parallel with bug fixing activities, probably needing similar resources.
- Increasing the generalizability of the proposed method by considering scheduling a larger set of bugs.
- Supplementing the scheduling with cost issues, i.e., the cost incurred in investing resources to perform different activities might impact the value objective.
- Analyzing the sensitivity of parameters in the GA and the fitness function, such as population size of the GA, priority preference weight and severity preference weight of the fitness function.

The next chapter, Chapter 10, presents a summary and a discussion of the thesis results, the thesis' conclusions and recommended future work.

Chapter 10

Discussion and conclusions

Below we first summarize the individual chapters and then discuss and conclude what they mean. Finally, we discuss future work.

10.1 Summary

The research questions (Section 1.4) in this thesis were targeted towards investigating multiple related themes, with a focus on using metaheuristic search-based techniques, for supporting the decision-making processes within software V&V activities. These themes related to:

- The possibility of analyzing software fault history as a measurement technique to predict future software reliability (Chapters 3–4).
- Using measures to support test phase efficiency (Chapter 5).
- Using measures to support assignment of resources to fix faults (Chapter 9).
- Using measures to classify fault-prone parts of the software from non fault-prone parts (Chapter 6).

The thesis also investigated two methodological research questions (RQ5 and RQ6), making up Chapters 7–8, using search-based techniques that were relevant to the task of software quality measurement. These two research questions were related to the use of resampling methods and feature subset selection methods.

Chapter 2 consolidated the existing evidence, in the software engineering literature, that comparatively evaluates the symbolic regression application of GP with other techniques. The results of this study provided evidence in support of symbolic regression using GP for software quality classification, software fault prediction and software reliability growth modeling in comparison with regression/machine learning techniques and other models. However, the study results were inconclusive in judging whether or not GP was an effective technique for software cost/effort/size estimation.

Chapter 3 presented an initial investigation into the predictive capabilities of applying symbolic regression using GP. The comparative evaluation with traditional software reliability growth models showed that symbolic regression using GP had a potential to be a valid fault prediction technique. Using three measures of model validity, the prequential likelihood ratio showed favorability for the GP models while the same was not the case with the Braun statistic and AMSE. The goodness of fit of the GP models was either equivalent or better in comparison with traditional models but not statistically significant in every case. The box plots of residuals and matched paired t-tests showed results in favor of GP models.

Chapter 4 carried forward the early positive results of using symbolic regression application of GP. The empirical investigation this time was into *cross-release* prediction of fault-count data from large and complex industrial and open-source software. The results were evaluated both quantitatively and qualitatively, while the comparisons were done with both machine-learning and traditional approaches to fault prediction. The results showed that, quantitatively, symbolic regression using GP was at least as competitive as other techniques for cross-release fault prediction. Qualitatively, symbolic regression using GP scored better for transparency of resulting solutions and generality, in comparison with comparative techniques. On the other hand, ease of configuration was found not to be a strength for symbolic regression using GP.

Chapter 5 presented an extensive empirical evaluation of various techniques for predicting the number of faults slipping through to the four test phases of unit, function, integration and system testing. A variety of techniques were found to be useful in such a prediction task, both in terms of predictive accuracy and goodness of fit. However, the group of search-based techniques *consistently* gave better predictions, having a representation at all the test phases. Human expert predictions were, however, among the better techniques at two of the four test phases. The chapter also reported on the results of a survey based on an industrial questionnaire that confirmed the usefulness of predicting the number of faults slipping through to the four test phases, particularly in terms of: (*i*) Baselining the expected effort to be spent on different test phases. (*ii*) Reducing the overall cost of testing by improving the test effort on phases with high fault-slippage. (*iii*) Visualizing the expected fault-detection efficiency of test phases. The survey results also highlighted the need for the prediction techniques to be part of an automated tool, the users being aware of the working principles of the techniques and the techniques being able to determine the form of the relationship between the inputs and the outputs.

Chapter 6 evaluated the use of faults-slip-through data as a potential predictor of fault-proneness at integration and system test levels for data gathered from two industrial projects. The performance of a variety of classifiers was assessed using AUC and the location of (PF, PD) pairs in the ROC space. The results showed that faults-slip-through data had the potential to be a generally useful predictor of fault-proneness at integration and system test levels. As for the different classifiers, GP performed consistently across both integration and system test levels in terms of AUC and the location of (PF, PD) pairs in the ROC space. The visualization of (PF, PD) pairs in the ROC space provided another opportunity for the test team to assess a classifier performance with respect to the perfect (PF, PD) pair of (0, 1). A distance metric could then be calculated, with different weights assigned to represent the misclassification costs of PF and PD, to select a classifier most suited for the project.

Chapter 7 reported on an extensive empirical comparison of five resampling methods using GP as a classifier over five public domain software data sets from the PROMISE data repository. Using (PF, PD) pair data across five data sets, bootstrapping gave results in the preferred region of the ROC space for two data sets, indicating that bootstrapping should be considered as a resampling method of choice in predictive studies in software engineering. However, where the statistical comparison of individual resampling methods was concerned, based on AUC, there were no significant differences. We then highlighted the reasons for such an outcome and attributed it to imbalanced data sets, insignificant predictor variables and high dimensional data sets.

Chapter 8 evaluated whether or not the use of feature subset selection (FSS) methods had any significant affect on the classification accuracy of software fault prediction, when used with two diverse learning algorithms: C4.5 and naïve Bayes. We compared a total of seven FSS methods, representing a mix of state of the art methods and an evolutionary computation method, on five software fault prediction data sets from the PROMISE data repository. Our findings showed that the use of these FSS methods do not lead to statistically significant differences in the classification accuracies (measured using AUC) for C4.5 and naïve Bayes. However, a smaller set of methods—CFS, WRP and GP—consistently selected fewer attributes without degrading classification accuracy.

Chapter 9 presented a search-based resource allocation method for bug fixing tasks. We proposed models for the bug fixing process, the human resources and the capability matching method between bug fixing activities and human resources. On the basis of these models and our proposed method, the resources were allocated for bug fixing activities using a genetic algorithm (GA). Depending on different objectives, three scenarios were discussed using an industrial data set, and the results showed that GA was able to provide schedules having balanced different objectives and, thus, providing maximum value for the organization. Comparison with hill-climbing showed that GA gave statistically better results in terms of maximizing the value objective.

10.2 Discussion

This thesis explores the synergy between search-based software engineering (SBSE) and predictive modeling for providing effective decision-support for V&V activities. We have shown that search-based techniques are scalable in the context of large software projects and, furthermore, that the results are consistently accurate. This also comes with an added advantage with the explanatory value of the results, i.e., these techniques are not black box approaches like the application of artificial neural networks to predictive modeling (discussed in Section 4.9). The identification of building blocks and progressively improving overall fitness (Section 3.4) enables search-based techniques to adapt to different operational situations. Additionally, since the search-based techniques are automated, to a large extent, once an expert completes the design of the search-based technique, the technique returns a near-optimal (or possibly an optimal solution) with respect to the fitness function used. This way the burden of search shifts from the human to the machine. However, this search needs not to be fully automated (giving rise to human-in-the-loop or interactive evolution [129]). There is an interesting trade-off in play here. Chapter 5 shows that on one hand, the human predictions regarding the number of faults slipping through to test phases are characterized by extreme fluctuations, while on the other hand, they are as accurate as the search-based techniques. (However, the bottom line being that the search-based techniques are always more *consistently* accurate.)

A synthesis of evidence on using GP for predictive modeling in software engineering (Chapter 2) reveals that the use of GP, as being a potentially effective tool in predictive modeling, has to be evaluated more on large-scale projects to increase generalizability. Moreover, the performance of search-based techniques needs to be compared against a larger set of more representative comparison groups. The synthesis also shows that the available evidence present an opportunity of improving the design of future studies (e.g., by explicitly stating the resampling strategy and the fitness function used). It is obvious, from reading Chapter 2, that research opportunities are plenty; both in terms of empirically evaluating the use of search-based techniques and carrying out methodological investigations to improve the design of predictive modeling studies.

The initial part of the thesis consists of empirical evaluations of search-based techniques as a replacement for more traditional software reliability growth models

(SRGMs) for estimating the future reliability of software. With the existence of a large number of SRGMs, and the accompanying assumptions they require to be fulfilled, a search-based technique like GP is an ideal replacement since GP evolution is not dependent on any assumptions and is a general-purpose technique. The consistently accurate performance of GP across multiple industrial data sets (Sections 3.5 and 3.6), attributed to the generation of progressively fitter solutions using building blocks (Section 3.4), means that GP is indeed a general purpose replacement of traditional SRGMs. Furthermore, an automated mechanism of cross-release prediction of faults is required, when considering the adoption of agile software development methodologies with short-timed releases. GP, in comparison with typical machine learning approaches and traditional SRGMs, comes out to be consistently accurate in this prediction task (Section 4.7). The use of GP in varying operational situations is, thus, validated.

We are mindful that while quantitative performance is of primary interest, certain qualitative aspects also play a part in accepting new techniques. Solution transparency and generality are added qualitative factors to go with the consistently accurate quantitative results given by GP (Section 4.9). Ease of configuring the GP parameters, on the other hand, is one qualitative factor that makes GP hard to use and represents interesting future work to undertake (Section 10.4).

The thesis, while further expanding the scope of applying search-based techniques, evaluates their use for predicting the number of faults slipping through to different test phases. In addition to being in an industrial context and dealing with large-scale projects, the comparison groups include human experts to investigate if search-based techniques are competitive (compared to engineers). It is shown that the group of search-based techniques consistently performs better than other techniques at all the test phases (Section 5.4.2). Human expert predictions are also among the better techniques at unit and integration test phases but are marked with extreme fluctuations for function and integration test phases. This shows that, in one way, the human predictions are limited in being consistently accurate, something that the search-based techniques excel in. The generality of the search-based techniques is therefore impressive in predicting the number of faults slipping through (FST) to each of the test phases. The results of predicting FST are then further evaluated in order to assess if FST can be used as an effective predictor of fault-proneness at integration and system test phases. This is unique since none of the previous studies on software quality classification focuses on identifying fault-prone software components at different test phases (Section 6.2) while making use of FST as a predictor of fault-proneness. The results shows that FST is indeed a useful predictor of fault-proneness and that among the different techniques, GP is able to show impressive AUC values (Section 6.6).

After having empirically evaluated the use of search-based techniques for a variety of research themes, the thesis focuses next on methodological investigations relevant for the design of predictive studies in software engineering. The first of these methodological investigations are targeted at comparing a variety of resampling methods to predict fault-prone software components using GP as a classifier. The results are interesting (Section 7.3.5), in that the various resampling methods did not differ significantly; however, the results also indicate that there are other contributing factors that have a synergetic effect on the outcome of software quality classification, e.g., imbalanced data sets, insignificant predictor variables and high dimensional data sets (Section 7.5). The other methodological investigation in this thesis is targeted at assessing the impact of different feature subset selection methods for software quality classification. The outcomes are interesting in that the use of a search-based technique like GP performs equally well in comparison with other state of the art feature subset selection methods. The strength for a search-based technique like GP is that feature selection is an implicit part of the GP evolution process and that the feature selection is performed without human bias. Finally, this thesis then also provides a search-based solution to triage and fix bugs by taking into account resource capability and availability. This presents an automated way of effectively triaging bugs by striking a balance between competing constraints.

10.3 Conclusions

As given in Section 1.1, the aim of this thesis was to evaluate the applicability of search-based techniques across a variety of research themes within software V&V. The context for the different studies in this thesis had been real software, representing both industrial and open-source projects. Having evaluated the use of search-based techniques across several research themes, and in varying degrees of contexts consisting of industrial and open-source software projects, the overall conclusion of the thesis is that the accuracy and consistency of search-based techniques makes them an ideal tool to use for improving input for decision-making, regarding software quality. Hence, this thesis has shown that the application of these techniques are well suited for solving practical problems within software V&V that are concerned with making predictions regarding software quality. The search-based techniques have shown to be more general purpose techniques, while not being dependent on assumptions common with other techniques. Being more flexible than their traditional counterparts, search-based techniques contribute towards automatic generation of relationships among process variables; that is highly desirable considering the ever-shrinking development cycles. Also, this thesis has provided early indications that the use of search-based techniques is indeed industrially scalable.

More specific conclusions, which can be drawn from this thesis, are given below:

1. The systematic review in Chapter 2 concluded that while there was evidence in software engineering literature in support of using symbolic regression application of GP for software quality classification, software fault prediction and software reliability growth modeling; the results were inconclusive for software cost/effort/size estimation (Section 2.4). Moreover, the symbolic regression application of GP was not dependent on unrealistic assumptions that were common in traditional SRGMs. Due to that reason, GP promised to be a valid tool in situations where different traditional models had inconsistent results. It was also concluded that the variety of comparison groups were represented poorly in studies reporting GP's application to predictive modeling in software engineering.

2. The Stage 1 study in Chapter 3 concluded that the evolutionary search mechanism of symbolic regression using GP was suitable for predicting the future software reliability in terms of number of faults (Section 3.4). This suitability was based on the identification of building blocks and progressively improving overall fitness. Based on this conclusion, the use of GP for predicting the future software reliability in an industrial context was deemed useful.

3. The Stage 2 study in Chapter 3 concluded that, based on the weekly fault count data from three different industrial software projects, the results for goodness of fit and prediction accuracy were statistically significant in favor of models built using symbolic regression application of GP (Section 3.5). This result gave early indications of the usefulness of GP as a prediction tool in an industrial context.

4. The Stage 3 study in Chapter 3 concluded that based on the comparative evaluation of models from symbolic regression application of GP and three traditional software reliability growth models, one out of three measures of model validity favored the GP models. The measures for goodness of fit and model bias showed that models built using symbolic regression application of GP were at least competitive to traditional software reliability growth models (Section 3.6). This was one of the early results that indicated GP to be at least as competitive as other techniques for predicting future reliability of software in an industrial context.

5. The quantitative evaluation of models built using symbolic regression application of GP in Chapter 4 showed that for cross-release fault predictions, they were at least as competitive as other machine learning and traditional models (Section 4.7). Hence, it was concluded that GP is a feasible prediction tool across different releases of software. This indicated that the development team could

use GP to make important decisions regarding the quality of their deliverables. GP models also showed a decent ability to adapt to different time spans of releases (on the basis of the different lengths of the testing sets for different data sets). The study showed that GP was least affected by moderate differences in the release durations and can predict decently with variable time units into future. Additionally, having evaluated the performance on diverse data sets from different application domains, it further pointed out the flexibility of GP, i.e., suited a variety of data sets.

6. Chapter 4 further concluded that there was a need to take into account qualitative factors for assessing the practical utility of a prediction system. The solutions given by the symbolic regression application of GP were open to interpretation but they might be complex and might not be able to give logical explanation of the relationships. Furthermore, the parameter tuning problem was time consuming and therefore ease of configuration was not a strength for symbolic regression application of GP (Sections 4.9 and 2.4).

7. Chapter 5 concluded that the group of search-based techniques were consistently better in predicting FST at each test phase. It was also concluded that the human predictions regarding the number of faults slipping through to various test phases could be well supported by the use of search-based techniques in an industrial context. The combination of human and automated search procedures (like any of the search-based techniques) has the potential to provide improved prediction results. The results of an industrial questionnaire concluded that the experts were in agreement that such predictions were useful and highlighted usability aspects that needed to be part of a multi-criteria based evaluation system (Section 5.6).

8. Another conclusion that could be drawn from Chapter 5 was that the search-based techniques (and other better performing techniques) performed well outside their respective training ranges, i.e., the predictions were evaluated for 15 weeks of an on-going project after being trained on another baseline project data. The over-fitting was within acceptable limits, and this was particularly encouraging considering the fact that we were dealing with large projects where the degree of variability in fault occurrences can be large. This issue was also related to the amount of data available for training the different techniques, which, in case of large projects is typically available.

9. Based on industrial data sets, Chapter 6 concluded that the faults-slip-through data had the potential to be a generally useful predictor of fault-proneness at integration and system test levels (Section 6.6). Moreover, the consistent performance of GP at the two test levels further validated the use of search-based

techniques as a software quality classification approach. Thus, collecting the FST data in large-scale industrial projects was beneficial, not only in improving the test level efficiency, but also in identifying fault-prone software components from the non-fault prone ones.

10. Chapter 7 concluded that bootstrapping should be considered as a resampling method of choice in predictive studies in software engineering. However, where the statistical comparison of individual resampling methods was concerned, based on AUC, there were no significant differences. This could be attributed to imbalanced data sets, insignificant predictor variables and high dimensional data sets. Hold-out validation performed less satisfactorily for comparatively smaller data sets where LOOCV performed better due to optimal use of the training data. For comparatively larger data sets 10-fold cross-validation was a better choice as compared to LOOCV (Section 7.4).

11. Chapter 8 concluded that the use of various feature subset selection methods do not lead to statistically significant differences in the classification accuracies (measured using AUC) for C4.5 and naïve Bayes. However, a smaller set of methods—CFS, WRP and GP—consistently selected fewer attributes without degrading classification accuracy (Section 8.5). Hence, the use of a search-based technique like GP was validated as a potentially useful FSS method.

12. Chapter 9 concluded that GA was able to provide schedules having balanced different objectives and brought maximum value for the organization. Compared to hill-climbing, GA gave statistically better results in terms of maximizing the value objective (Sections 9.6 and 9.7).

10.4 Future research

Through empirical investigations and the literature review conducted as part of this thesis, we anticipate a promising future where there are further research opportunities for evaluating the application of search-based techniques in software verification and validation. Even though the end of each chapter in this thesis mention future work, the following list highlights the most important future research opportunities. These research opportunities are grouped into three different themes (scope expansion, algorithmic enhancements and design enhancements):

- Scope expansion

  - Empirically evaluate the effectiveness of using search-based techniques in prediction across other phases of the software development life cycle, such as e.g. maintenance task effort.
  - Empirically evaluate the use of other search-based approaches for predictions, such as particle swarm optimization and a multi-criteria based GP approach.
  - A general multi-criteria based evaluation system needs to be benchmarked that captures both the quantitative and the qualitative aspects of prediction techniques.
  - Empirically compare the faults-slip-through metric with other commonly used predictors of fault proneness to quantify any differences.
  - Empirically investigating the outcome of variants of bootstrapping for different software engineering data sets.

- Algorithmic enhancements

  - Empirically evaluate the use of different fitness functions to better guide search of feasible solutions for the search-based techniques.
  - Investigate the mechanisms of finding compact and less complex GP solutions.
  - Investigate the potential of saving the state information during GP evolved solutions so as to enhance the predictive accuracy on time-series nature of data.
  - Study the impact of different settings of GP parameters versus the resampling methods so that one can assess how much variability in the outcome can be attributed to each one of them.
  - Compare a multi-criteria GP fitness function for feature subset selection, e.g., one that combines cost of classification with accuracy.

- Design enhancements

  - Investigate the potential of adaptive parameter control during a GP run to ease parameter tuning for the GP algorithm.
  - Evaluate the adaptive capability of the GP algorithm for different sets of independent software metrics.
  - Evaluate the effectiveness of GP predictions at finer levels of detail by collecting different metrics at the code and/or module level.

- Design of a probabilistic model on how likely different fault counts or slips in earlier phases are for predicting fault-proneness in later phases.

Moreover, the above mentioned future work require developing tools to help engineers and managers in using search-based approaches in their day to day activities (e.g., implementing functions in SAS and developing toolboxes for MATLAB). Last but not the least, there are further opportunities to evaluate existing applications of search-based techniques in particular domains and, especially, in ongoing projects in industry.

References

[1] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood. Evaluation of competing software reliability predictions. *IEEE Transactions on Reliability*, 12(9):950–967, 1986.

[2] A. Abraham. Real time intrusion prediction, detection and prevention programs. In *Proceedings of the 2008 IEEE International Conference on Intelligence and Security Informatics (ISI'08)*, Piscataway, NJ, USA, 2008.

[3] W. Adnan, M. Yaakob, R. Anas, and M. Tamjis. Artificial neural network for software reliability assessment. In *Proceedings of the IEEE Trends and Developments in Converging Technology towards 2020 (TENCON'00)*, 2000.

[4] W. Afzal. Using faults-slip-through metric as a predictor of fault-proneness. In *Proceedings of the 21st Asia Pacific Software Engineering Conference (APSEC'10)*. IEEE, 2010.

[5] W. Afzal and R. Torkar. On the application of genetic programming for software engineering predictive modeling: A systematic review. In print: Expert Systems With Applications, accessible from: http://iaser.tek.bth.se/torkar/ESWA-S-10-00162.pdf.

[6] W. Afzal and R. Torkar. A comparative evaluation of using genetic programming for predicting fault count data. In *Proceedings of the 3rd International Conference on Software Engineering Advances (ICSEA'08)*, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[7] W. Afzal and R. Torkar. Lessons from applying experimentation in software engineering prediction systems. In *Proceedings of the 2nd International Workshop on Software Productivity Analysis And Cost Estimation (SPACE'08), co-located with Asia Pacific Software Engineering Conference (APSEC'08)*, 2008.

[8] W. Afzal and R. Torkar. Suitability of genetic programming for software reliability growth modeling. In *Proceedings of the 1st International Symposium on Computer Science and its Applications (CSA'08)*, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[9] W. Afzal, R. Torkar, and R. Feldt. Prediction of fault count data using genetic programming. In *Proceedings of the 12th IEEE International Multitopic Conference (INMIC'08)*. IEEE, 2008.

[10] W. Afzal, R. Torkar, and R. Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[11] W. Afzal, R. Torkar, R. Feldt, and T. Gorschek. Genetic programming for cross-release fault count predictions in large and complex software projects. In Monica Chis, editor, *Evolutionary Computation and Optimization Algorithms in Software Engineering: Applications and Techniques*, pages 94–126. IGI Global, Hershey, USA, 2009.

[12] W. Afzal, R. Torkar, R. Feldt, and G. Wikstrand. Search-based prediction of fault-slip-through in large software projects. In *Proceedings of the 2nd International Symposium on Search-Based Software Engineering (SSBSE'10)*. IEEE Computer Society, 2010.

[13] J. S. Aguilar-Ruiz, I. Ramos, J. C. Riquelme, and M. Toro. An evolutionary approach to estimating software development projects. *Information and Software Technology*, 43(14):875 – 882, 2001.

[14] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[15] F. Akiyama. An example of software system debugging. *International Federation for Information Processing Congress*, 71(1):353–359, 1971.

[16] J. T. Alander. An indexed bibliography of genetic programming. Technical Report 94-1-GP, Department of Information Technology and Industrial Management, University of Vaasa, 1995.

[17] E. Alba and J. Chicano. Software project management with GAs. *Information Sciences*, 177(11):2380–2401, 2007.

[18] E. Alfaro-Cid, E. W. McGookin, D. J. Murray-Smith, and T. I. Fossen. Genetic programming for the automatic design of controllers for a surface ship. *IEEE Transactions on Intelligent Transportation Systems*, 9(2):311–321, 2008.

[19] S. Aljahdali, A. Sheta, and D. Rine. Prediction of software reliability: A comparison between regression and neural network non-parametric models. In *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'01)*, 2001.

[20] E. Alpaydin. *Introduction to machine learning*. The MIT Press, 2004.

[21] P. L. Alreck and R. B. Settle. *The survey research handbook: Guidelines and strategies for conducting a survey*. IRWIN Professional Publishing, 1995.

[22] B. Andersson, P. Svensson, P. Nordin, and M. Nordahl. Reactive and memory-based genetic programming for robot control. In *Proceedings of the 2nd European Workshop on Genetic Programming (EuroGP'99)*, Berlin, Germany, 1999.

[23] F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, February 1973.

[24] G. Antoniol, M. Di Penta, and M. Harman. Search-based techniques applied to optimization of project planning for a massive maintenance project. In *Proceedings of the 21st IEEE Interantional Conference on Software Maintenance (ICSM'05)*. IEEE, 2005.

[25] J. Anvik, L. Hiew, and G. Murphy. Who should fix this bug? In *Proceedings of the International Conference on Software Engineering (ICSE'06)*, 2006.

[26] E. Arisholm, L. C. Briand, and E. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

[27] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Evolutionary computation 1 – Basic algorithms and operators*. Taylor & Francis Group, LLC, 27 Madison Avenue, New York, USA, 2000.

[28] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The next release problem. *Information and Software Technology*, 43(14):883– 890, 2001.

[29] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic programming - An introduction*. Morgan Kaufmann Publishers, Inc., 1998.

[30] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende, and W. R. S. Junior. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, 1995.

[31] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[32] D. Berger. Introduction to resampling techniques. `http://wise.cgu.edu/downloads/Introduction%to%Resampling%Techniques%060420.doc`, 2011. Last checked: 15 March 2011.

[33] A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[34] B. W. Boehm. *Software engineering economics*. Prentice-Hall, Upper Saddle River, NJ, USA, 1981.

[35] B. W. Boehm. Industrial software metrics top 10 list. *IEEE Software*, 4(9):84–85, 1987.

[36] B. W. Boehm and V. R. Basili. Software defect reduction top 10 list. *Computer*, 34(1):135–137, 2001.

[37] G. Boetticher, T. Menzies, and T. Ostrand. PROMISE repository of empirical software engineering data, 2007. `http://promisedata.org/`, Last checked: 20 February 2011, West Virginia university, Department of computer science.

[38] S. Bouktif, G. Antoniol, E. Merlo, and M. Neteler. A novel approach to optimize clone refactoring activity. In *Proceedings of the 2006 Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, New York, NY, USA, 2006. ACM.

[39] S. Bouktif, H. Sahraoui, and G. Antoniol. Simulated annealing for improving software quality prediction. In *Proceedings of the 2006 Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, New York, NY, USA, 2006. ACM.

[40] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for experimenters: An introduction to design, data analysis, and model building*. Wiley-Interscience, 1978.

[41] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[42] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[43] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[44] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571 – 583, 2007.

[45] L. C. Briand, V. R. Basili, and C. J. Hetmanski. Developing interpretable models with optimized set reduction for identifying high-risk software components. *IEEE Transactions on Software Engineering*, 19(11):1028–1044, 1993.

[46] L. C. Briand, V. R. Basili, and W. M. Thomas. A pattern recognition approach for software engineering data analysis. *IEEE Transactions on Software Engineering*, 18(11):931–942, 1992.

[47] L. C. Briand, K. El-Emam, B. Freimut, and O. Laitenberger. A comprehensive evaluation of capture-recapture models for estimating software defect content. *IEEE Transactions on Software Engineering*, 26(6), 2000.

[48] L. C. Briand, K. El-Emam, and S. Morasca. On the application of measurement theory in software engineering. *Empirical Software Engineering*, 1(1):61–88, 1996.

[49] L. C. Briand, W. L. Melo, and J. Wüst. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering*, 28(7):706–720, 2002.

[50] L. Brieman and P. Spector. Submodel selection and evaluation in regression: The x-random case. *International statistical review*, 60(3):291–319, 1992.

[51] S. Brocklehurst and B. Littlewood. *Handbook of software reliability engineering, Editor M. R. Lyu*, chapter 4: Techniques for prediction analysis and recalibration, pages 119–166. McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.

[52] W. D. Brooks and R. W. Motley. Analysis of discrete software reliability models. Technical Report ADA086334, IBM Federal Systems, 1980.

[53] J. Brownlee. WEKA plug-in for AIRS. `http://wekaclassalgos.sourceforge.net/`, 2011. Last checked: 15 March 2011.

[54] R. D. Buck and J. H. Dobbins. Application of software inspection methodology in design and code. In *Proceedings of the Symposium on Software Validation: Inspection-Testing-Verification-Alternatives*, New York, NY, USA, 1984. Elsevier North-Holland, Inc.

[55] Bugzilla – A bug tracking tool. `http://www.bugzilla.org/`. Last checked: 21 March 2010.

[56] C. J. Burgess and M. Lefley. Can genetic programming improve software effort estimation? A comparative evaluation. *Information and Software Technology*, 43(14):863–873, 2001.

[57] E. K. Burke and G. Kendall, editors. *Search methodologies – Introductory tutorials in optimization and decision support techniques*. Springer Science and Business Media, Inc., 233 Spring Street, New York, USA, 2005.

[58] L. I. Burke. Introduction to artificial neural systems for pattern recognition. *Computers & Operations Research*, 18(2), 1991.

[59] K. Y. Cai, L. Cai, W. D. Wang, Z. Y. Yu, and D. Zhang. On the neural network approach in software reliability modeling. *Journal of Systems and Software*, 58(1):47–62, 2001.

[60] K. Y. Cai, C. Wen, and M. Zhang. A critical review on software reliability modeling. *Reliability Engineering and System Safety*, 32(3):357–371, 1991.

[61] K. Y. Cai, C. Wen, and M. Zhang. A novel approach to software reliability modeling. *Microelectronics and Reliability*, 33(15):2265–2267, 1993.

[62] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM symposium on Applied computing (SAC'06)*. ACM, 2006.

[63] S. Canu, Y. Grandvalet, V. Guigue, and A. Rakotomamonjy. SVM and kernel methods toolbox. Perception Systémes et Information, INSA de Rouen, Rouen, France, 2005.

[64] F. Castillo, A. Kordon, G. Smits, B. Christenson, and D. Dickerson. Pareto front genetic programming parameter selection based on design of experiments and industrial data. In *Proceedings of the 2006 Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, New York, USA, 2006. ACM.

[65] C. Catal and B. Diri. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences*, 179(8):1040 – 1058, 2009.

[66] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346 – 7354, 2009.

[67] V. U. B. Challagulla, F. B. Bastani, I. Yen, and R. A. Paul. Empirical assessment of machine learning based software defect prediction techniques. In *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05)*, Washington, DC, USA, 2005. IEEE Computer Society.

[68] C. Chang, H. Jiang, Y. Di, D. Zhu, and Y. Ge. Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology*, 50(11):1142–1154, 2008.

[69] Z. Chen, B. W. Boehm, T. Menzies, and D. Port. Finding the right data for software cost modeling. *IEEE Software*, 22(6):38–46, 2005.

[70] J. G. Cleary and L. E. Trigg. K*: An instance-based learner using an entropic distance measure. In *Proceedings of the 12th International Conference on Machine Learning (ICML'95)*, 1995.

[71] B. T. Compton and C. Withrow. Prediction and control of Ada software defects. *Journal of Systems and Software*, 12(3):199–207, 1990.

[72] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software engineering metrics and models*. Benjamin/Cummings, 1986.

[73] E. O. Costa, G. A. de Souza, A. Pozo, and S. Vergilio. Exploring genetic programming and boosting techniques to model software reliability. *IEEE Transactions on Reliability*, 56(3):422–434, 2007.

[74] E. O. Costa and A. Pozo. A mu + lambda – GP algorithm and its use for regression problems. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, Washington, DC, USA, 2006. IEEE Computer Society.

[75] E. O. Costa, S. Vergilio, A. Pozo, and G. Souza. Modeling software reliability growth with genetic programming. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*, Washington, DC, USA, 2005. IEEE Computer Society.

[76] J. Crespo, J. J. Cuadrado, L. Garcia, O. Marban, and M. I. Sanchez-Segura. Survey of artificial intelligence methods on software development effort estimation. In *Proceedings of the 10th ISPE International Conference on Concurrent Engineering (ICCE'03)*. Swets en Zeitlinger B.V., 2003.

[77] J. W. Creswell. *Research design – Qualitative, quantitative and mixed method approaches*. Sage Publications, United Kingdom/India, second edition, 2003.

[78] D. Cubranic and G. Murphy. Automatic bug triage using text categorization. In *Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE'04)*, 2004.

[79] L-Ola Damm. *Early and cost-effective software fault detection – Measurement and implementation in an industrial setting*. PhD thesis, Blekinge Institute of Technology, 2007.

[80] L-Ola Damm, L. Lundberg, and C. Wohlin. Faults-slip-through–A concept for measuring the efficiency of the test process. *Software Process: Improvement & Practice*, 11(1), 2006.

[81] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(1-4):131–156, 1997.

[82] M. A. de Almeida, H. Lounis, and W. L. Melo. An investigation on the use of machine learned models for estimating correction costs. In *Proceedings of the 20th International Conference on Software Engineering (ICSE'98)*, 1998.

[83] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[84] G. Denaro and M. Pezze. An empirical evaluation of fault-proneness models. In *Proceedings of the 24th International Conference on Software Engineering (ICSE'02)*, 2002.

[85] J. M. Desharnais. Analyse statistique de la productivité des projets de développement en informatique à partir de la technique des points de fonction. Master's thesis, University of Montreal, Montreal, Canada, 1989.

[86] T. Dohi, Y. Nishio, and S. Osaki. Optimal software release scheduling based on artificial neural networks. *Annals of Software Engineering*, 8(1-4):167–185, 1999.

[87] J. J. Dolado. A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, 26(10), 2000.

[88] J. J. Dolado. On the problem of the software cost function. *Information and Software Technology*, 43(1):61 – 72, 2001.

[89] J. J. Dolado and L. Fernandez. Genetic programming, neural networks and linear regression in software project estimation. In *Proceedings of the International Conference on Software Process Improvement, Research, Education and Training (INSPIRE'98)*, London, 1998. British Computer Society.

[90] J. J. Dolado, L. Fernandez, M. C. Otero, and L. Urkola. Software effort estimation: The elusive goal in project management. In *Proceedings of the International Conference on Enterprise Information Systems (ICEIS'99)*, 1999.

[91] T. Dybå, T. Dingsøyr, and G. K. Hanssen. Applying systematic reviews to diverse study types: An experience report. In *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM'07)*, 2007.

[92] T. Dybå, B. A. Kitchenham, and M. Jørgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):58–65, 2005.

[93] B. Efron and G. Gong. A leisurely look at the bootstrap, the jackknife, and cross-validation. *The American Statistician*, 37(1):36–48, 1983.

[94] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC Press, USA, 1998.

[95] K. El-Emam, S. Benlarbi, N. Goel, and S. N. Rai. Comparing case-based reasoning classifiers for predicting high risk software components. *Journal of Systems and Software*, 55(3):301–320, 2001.

[96] M. Evett, T. M. Khoshgoftar, P. Chien, and E. Allen. GP-based software quality prediction. In *Proceedings of the 3rd Annual Genetic Programming Conference (GP'98)*, 1998.

[97] W. Farr. SMERFS3 homepage, 2009. `http://www.slingcode.com/smerfs/downloads/`, Last checked: 05 March 2009.

[98] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[99] R. Feldt and P. Nordin. Using factorial experiments to evaluate the effect of genetic programming parameters. In *Proceedings of the European Conference on Genetic Programming (EuroGP'00)*, London, UK, 2000. Springer-Verlag.

[100] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999.

[101] N. E. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, 26(8):797–814, 2000.

[102] N. E. Fenton and S. L. Pfleeger. *Software metrics: A rigorous and practical approach*. Course Technology, Boston, MA, USA, 2nd edition, 1998.

[103] C. Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13(2), 2001.

[104] W. A. Florac, R. Park, and A. D. Carleton. Practical software measurement: Measuring for process management and improvement, handbook. Technical Report CMU/SEI-97-HB-003, DoD, Data analysis centre for software, 1997.

[105] T. Foss, E. Stensrud, B. A. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering*, 29(11), 2003.

[106] J. E. Gaffney. Estimating the number of faults in code. *IEEE Transactions on Software Engineering*, 10(4):459–465, 1984.

[107] K. Gao and T. M. Khoshgoftaar. A comprehensive empirical study of count models for software fault prediction. *IEEE Transactions on Reliability*, 56(2), 2007.

[108] A. L. Goel. Software reliability models: Assumptions, limitations, and applicability. *IEEE Transactions on Software Engineering*, SE-11(12):1411–1423, 1985.

[109] A. L. Goel and K. Okumoto. Time dependent error detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, R-28(3):206–211, 1979.

[110] G. Gong. Cross-validation, the jackknife, and the bootstrap: Excess error estimation in forward logistic regression. *Journal of the American Statistical Association*, 81(393):108–113, 1986.

[111] P. I. Good. *Resampling methods—A practical guide to data analysis*. Birkhäuser, USA, 2006.

[112] R. B. Grady. *Practical software metrics for project management and process improvement*. Prentice-Hall, Upper Saddle River, NJ, USA, 1992.

[113] A. Gray and S. MacDonell. A comparison of techniques for developing predictive models of software metrics. *Information and Software Technology*, 39(6):425 – 437, 1997.

[114] M. Green and M. Ohlsson. Comparison of standard resampling methods for performance estimation of artificial neural network ensembles. In *Proceedings of the 3rd International Conference on Computational Intelligence in Medicine and Healthcare (CIMED'07)*, Plymouth, England, 2007.

[115] S. R. Gunn. Support vector machines for classification and regression. Technical report, School of electronics and computer science, University of Southampton, 1998.

[116] L. Guo, Y. Ma, B. Cukic, and H. Singh. Robust prediction of fault-proneness by random forests. In *Proceedings of the 2004 International Symposium on Software Reliability Engineering (ISSRE'04)*. IEEE, 2004.

[117] P. Guo and M. R. Lyu. A pseudoinverse learning algorithm for feedforward neural networks with stacked generalization applications to software reliability growth data. *Neurocomputing*, 56:101–121, 2004.

[118] N. Gupta and M. P. Singh. Estimation of software reliability with execution time model using the pattern mapping technique of artificial neural network. *Computers & Operations Research*, 32(1):187–199, 2005.

[119] R. Gutierrez-Osuna. Lecture notes on cross-validation. `http://research.cs.tamu.edu/prism/lectures/pr/pr_l13.pdf`, 2011. Last checked: 15 March 2011.

[120] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[121] T. Gyimóthy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005.

[122] M. A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the International Conference on Machine Learning (ICML'00)*, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[123] M. A. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[124] M. A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1437–1447, 2003.

[125] M. H. Halstead. *Elements of software science*. Elsevier, North-Holland, 1977.

[126] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *RADIOLOGY*, 143(1):29–36, 1982.

[127] M. Harman. Search based software engineering. In *Proceedings of the Workshop on Computational Science in Software Engineering, collocated with 6th International Computational Science (ICCS'06)*, Berlin, Germany, 2006. Lecture Notes in Computer Science Vol. 3994.

[128] M. Harman. The current state and future of search-based software engineering. In *Proceedings of Future of Software Engineering at 29th International Conference on Software Engineering (FOSE'07)*. IEEE Computer Society, USA, 2007.

[129] M. Harman. The relationship between search based software engineering and predictive modeling. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE'10)*, New York, NY, USA, 2010. ACM.

[130] M. Harman and J. Clark. Metrics are fitness functions too. In *Proceedings of the 10th International Symposium on Software Metrics (METRICS'04)*. IEEE, 2004.

[131] M. Harman and B. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833 – 839, 2001.

[132] M. Harman, S. A. Mansouri, and Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03, Department of Computer Science, King's College London, 2009.

[133] E. Hart, P. Ross, and D. Corne. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines*, 6(2):191–220, 2005.

[134] K. Henningsson. *A fault classification approach to software process improvement*. Blekinge Institute of Technology Licentiate Series No. 2005:03, Ronneby, Sweden, 2005.

[135] S. Ho, M. Xie, and T. Goh. A study of the connectionist models for software reliability prediction. *Computers and Mathematics with Applications*, 46(3):1037–1045, 2003.

[136] J. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[137] M. Hollander and D. A. Wolfe. *Non-parametric statistical methods*. John Wiley and Sons, Inc., 1999.

[138] L. Hribar. Implementation of FST in design phase of the project. In *Proceedings of the 31st Jubilee International Convention (MIPRO'08), Ericsson Nikola Tesla*, 2008.

[139] R. T. Hughes. Expert judgement as an estimating method. *Information and Software Technology*, 38(2):67–75, 1996.

[140] IEEE, Inc., USA. *IEEE standard classification for software anomalies, IEEE Standard 1044-1993*, 1993.

[141] The Institute of Electrical and Electronics Engineers, Inc., Los Alamitos, California. *Guide to the software engineering body of knowledge (SWEBOK ®)*, 2004.

[142] J. P. A. Ioannidis. Why most published research findings are false. *PLoS Medicine*, 2(8):696–701, August 2005.

[143] ISBSG. The International Software Benchmarking Standards Group Limited. http://www.isbsg.org/, 2009. Last checked: 18 March 2009.

[144] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[145] A. Janecek, W. Gansterer, M. Demel, and G. Ecker. On the relationship between feature selection and classification accuracy. In *Proceedings of the 3rd Workshop on New Challenges for Feature Selection in Data Mining and Knowledge Discovery (FSDM'08)*, Brookline, MA, USA, 2008. Microtome Publishing.

[146] Z. Jelinski and P. B. Moranda. Software reliability research. In *Statistical Computer Performance Evaluation, Editor W. Freiberger*. Academic Press, New York, USA, 1972.

[147] G. K. Jha, P. Thulasiraman, and R. K. Thulasiram. PSO based neural network for time series forecasting. In *International Joint Conference on Neural Networks (IJCNN'09)*, 2009.

[148] Y. Jiang, B. Cukic, and T. Menzies. Fault prediction using early lifecycle data. In *Proceedings of the The 18th IEEE International Symposium on Software Reliability Engineering (ISSRE'07)*, Washington, DC, USA, 2007. IEEE Computer Society.

[149] Y. Jiang, B. Cukic, and T. Menzies. PC1_req, JM1_req and CM1_req data sets. `http://promisedata.org/?cat=184`, 2011. Last checked: 15 March 2011.

[150] Y. Jiang, B. Cukic, T. Menzies, and N. Bartlow. Comparing design and code metrics for software quality prediction. In *Proceedings of the 4th international workshop on predictor models in software engineering (PROMISE'08)*, New York, NY, USA, 2008. ACM.

[151] E. Johannessen. Data mining techniques, candidate measures and evaluation methods for building practically useful fault-proneness prediction models. Master's thesis, University of Oslo, Department of Informatics, Oslo, Norway, 2008.

[152] C. Jones. The pragmatics of software process improvements, 1996. Software Process Newsletter of the Software Engineering Technical Council Newsletter, No. 5.

[153] M. Jørgensen. Experience with the accuracy of software maintenance task effort prediction models. *IEEE Transactions on Software Engineering*, 21(8):674–681, 1995.

[154] M. Jørgensen. BESTweb, a repository of software cost and effort estimation papers – Simula research laboratory. http://home.simula.no/BESTweb/, 2009. Last checked: 07 March 2009.

[155] M. Jørgensen, T. Dybå, and B. A. Kitchenham. Teaching evidence-based software engineering to university students. In *Proceedings of the 11th IEEE International Symposium on Software Metrics (METRICS'05)*, 2005.

[156] M. Jørgensen, G. Kirkebøen, D. I. K. Sjøberg, B. Anda, and L. Bratthall. Human judgement in effort estimation of software projects. In *Proceedings of the Workshop on Using Multi-disciplinary Approaches in Empirical Software Engineering Research, co-located with International Conference on Software Engineering (ICSE'00)*, Limerick, Ireland, 2000.

[157] M. Jørgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, 33(1):33–53, 2007.

[158] N. Juristo and A. M. Moreno. *Basics of software engineering experimentation*. Kluwer Academic Publishers, 2001.

[159] S. K. Kachigan. *Statistical analysis – An interdisciplinary introduction to univariate and multivariate methods*. Radius Press, 1982.

[160] K. Kaminsky and G. Boetticher. Building a genetically engineerable evolvable program (GEEP) using breadth-based explicit knowledge for predicting software defects. In *Proceedings of the 2004 IEEE Annual Meeting of the Fuzzy Information Processing (NAFIPS'04).*, 2004.

[161] K. Kaminsky and G. Boetticher. Defect prediction using machine learners in an implicitly data starved domain. In *Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (WMSCI'04)*, Orlando, FL, 2004.

[162] S. H. Kan. *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[163] N. Karunanithi. A neural network approach for software reliability growth modeling in the presence of code churn. In *Proceedings of the 4th International Symposium on Software Reliability Engineering (ISSRE'93)*, 1993.

[164] N. Karunanithi and Y. K. Malaiya. *Handbook of software reliability engineering, Editor M. R. Lyu*, chapter 17: Neural networks for software reliability engineering, pages 699–728. McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.

[165] N. Karunanithi, Y. K. Malaiya, and D. Whitley. Prediction of software reliability using neural networks. In *Proceedings of the 1991 International Symposium on Software Reliability Engineering (ISSRE'91)*, 1991.

[166] N. Karunanithi, D. Whitley, and Y. K. Malaiya. Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering*, 18(7):563–574, 1992.

[167] N. Karunanithi, D. Whitley, and Y. K. Malaiya. Using neural networks in reliability prediction. *IEEE Software*, 9(4):53–59, 1992.

[168] T. M. Khoshgoftaar and E. Allen. Logistic regression modeling of software quality. *International Journal of Reliability, Quality and Safety Engineering*, 6(4):303–317, 1999.

[169] T. M. Khoshgoftaar, E. Allen, J. Hudepohl, and S. Aud. Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks*, 8(4), 1997.

[170] T. M. Khoshgoftaar, E. Allen, W. D. Jones, and J. Hudepohl. Classification tree models of software quality over multiple releases. In *Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE'99)*, Washington, USA, 1999. IEEE Computer Society.

[171] T. M. Khoshgoftaar, E. Allen, W. D. Jones, and J. Hudepohl. Classification tree models of software quality over multiple releases. In *Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE'99)*, Washington, DC, USA, 1999. IEEE Computer Society.

[172] T. M. Khoshgoftaar and Y. Liu. A multi-objective software quality classification model using genetic programming. *IEEE Transactions on Reliability*, 56(2):237–245, 2007.

[173] T. M. Khoshgoftaar, Y. Liu, and N. Seliya. Module-order modeling using an evolutionary multi-objective optimization approach. In *Proceedings of the 10th International Symposium on Software Metrics (METRICS'04)*, Washington, DC, USA, 2004. IEEE Computer Society.

[174] T. M. Khoshgoftaar, Y. Liu, and N. Seliya. A multiobjective module-order model for software quality enhancement. *IEEE Transactions on Evolutionary Computation*, 8(6):593–608, 2004.

[175] T. M. Khoshgoftaar and J. Munson. Predicting software development errors using software complexity metrics. *IEEE Journal on Selected Areas in Communications*, 8(2):253–261, 1990.

[176] T. M. Khoshgoftaar, J. Munson, B. B. Bhattacharya, and G. D. Richardson. Predictive modeling techniques of software quality from software measures. *IEEE Transactions on Software Engineering*, 18(11):979–987, 1992.

[177] T. M. Khoshgoftaar, A. Pandya, and H. More. A neural network approach for predicting software development faults. In *Proceedings of the 3rd International Symposium on Software Reliability Engineering (ISSRE'92)*, 1992.

[178] T. M. Khoshgoftaar and N. Seliya. Tree-based software quality estimation models for fault prediction. In *Proceedings of the 8th International Symposium on Software Metrics (METRICS'02)*, Washington, DC, USA, 2002. IEEE Computer Society.

[179] T. M. Khoshgoftaar and N. Seliya. Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering*, 8(3):255–283, 2004.

[180] T. M. Khoshgoftaar, N. Seliya, and Y. Liu. Genetic programming-based decision trees for software quality classification. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)*, 2003.

[181] T. M. Khoshgoftaar, N. Seliya, and N. Sundaresh. An empirical study of predicting software faults with case-based reasoning. *Software Quality Control*, 14(2):85–111, 2006.

[182] T. M. Khoshgoftaar and R. Szabo. Using neural networks to predict software faults during testing. *IEEE Transactions on Reliability*, 45(3):456–462, 1996.

[183] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Proceedings of the 9th International Workshop on Machine Learning (ML'92)*, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[184] N. R. Kiran and V. Ravi. Software reliability prediction by soft computing techniques. *Journal of Systems and Software*, 81(4), 2008.

[185] C. Kirsopp and M. Shepperd. Case and feature subset selection in case-based software project effort prediction. In *Proceedings of the 22nd SGAI International Conference on Knowledge-Based Systems and Applied Artificial Intelligence*, 2002.

[186] C. Kirsopp and M. Shepperd. Making inferences with small numbers of training sets. *IEE Proceedings Software*, 149(5):123–130, 2002.

[187] C. Kirsopp, M. Shepperd, and J. Hart. Search heuristics, case-based reasoning and software project effort prediction. In *Proceedings of the 2002 Annual Genetic and Evolutionary Computation Conference (GECCO'02)*, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[188] B. A. Kitchenham. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-001, Keele University, School of Computer Science and Mathematics, 2007.

[189] B. A. Kitchenham, T. Dybå, and M. Jørgensen. Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, Washington, DC, USA, 2004. IEEE Computer Society.

[190] B. A. Kitchenham and E. Mendes. Why comparative effort prediction studies may be invalid. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering (PROMISE'09)*, New York, NY, USA, 2009. ACM.

[191] B. A. Kitchenham, E. Mendes, and G. H. Travassos. Cross versus within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering*, 33(5):316–329, 2007.

[192] B. A. Kitchenham, L. M. Pickard, and S. J. Linkman. An evaluation of some design metrics. *Software Engineering Journal*, 5(1):50–58, 1990.

[193] B. A. Kitchenham, L. M. Pickard, S. MacDonell, and M. Shepperd. What accuracy statistics really measure? *IEE Proceedings Software*, 148(3), Jun 2001.

[194] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[195] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[196] I. Kononenko. Estimating attributes: Analysis and extensions of RELIEF. In *Proceedings of the European conference on Machine Learning (ECML'94)*, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[197] A. Kordon, G. Smits, E. Jordaan, and E. Rightor. Robust soft sensors based on integration of genetic programming, analytical neural networks, and support vector machines. In *Proceedings of the IEEE International Conference on E-Commerce Technology (CEC'02)*, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[198] M. Kotanchek, G. Smits, and A. Kordon. *Industrial strength genetic programming*. Kluwer, 2003.

[199] S. Kotsiantis, I. Zaharakis, and P. Pintelas. Machine learning: A review of classification and combining techniques. *Artificial Intelligence Review*, 26(3):159 – 190, 2007.

[200] J. R. Koza. *Genetic programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.

[201] W. B. Langdon and B. F. Buxton. Genetic programming for mining DNA chip data from cancer patients. *Genetic Programming and Evolvable Machines*, 5(3):251–257, 2004.

[202] W. B. Langdon, S. Gustafson, and J. R. Koza. The genetic programming bibliography. http://www.cs.bham.ac.uk/∼wbl/biblio/, 2009. Last checked: 13 February 2009.

[203] W. B. Langdon, R. Poli, N. F. McPhee, and J. R. Koza. Genetic programming: An introduction and tutorial, with a survey of techniques and applications. In J. Fulcher and L. C. Jain, editors, *Computational Intelligence: A Compendium*, volume 115 of *Studies in Computational Intelligence (SCI)*, chapter 22, pages 927–1028. Springer-Verlag, 2008.

[204] F. Lanubile and G. Visaggio. Evaluating predictive quality models derived from software measures: Lessons learned. *Journal of Systems and Software*, 38(3):225 – 234, 1997.

[205] P. Laplante and N. Ahmad. Pavlov's bugs: Matching repair policies with rewards. *IT Professional*, 11(4), 2009.

[206] N. Lavesson and P. Davidsson. Generic methods for multi-criteria evaluation. In *Proceedings of the SIAM International Conference on Data Mining (SDM'08)*, 2008.

[207] B. LeBaron and A. S. Weigend. A bootstrap evaluation of the effect of data splitting on financial time series. *IEEE Transactions on Neural Networks*, 9(1):213–220, 1998.

[208] M. Lefley and M. Shepperd. Using genetic programming to improve software effort estimation based on general data sets. In *Proceedings of the 2003 Annual Conference on Genetic and Evolutionary Computation (GECCO'03)*. ACM, 2003.

[209] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[210] P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam. Empirical evaluation of defect projection models for widely-deployed production software systems. In *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'04/FSE-12)*, New York, NY, USA, 2004. ACM.

[211] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

[212] C. X. Ling, J. Huang, and H. Zhang. AUC: A statistically consistent and more discriminating measure than accuracy. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.

[213] M. Lipow. Number of faults per line of code. *IEEE Transactions on Software Engineering*, 8(4):437–439, 1982.

[214] H. Liu and R. Setiono. A probabilistic approach to feature selection—A filter solution. In *Proceedings of International Conference on Machine Learning (ICML'96)*, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[215] Y. Liu and T. M. Khoshgoftaar. Genetic programming model for software quality classification. In *Proceedings of the 6th IEEE International Symposium on High-Assurance Systems Engineering (HASE'01)*, Washington, DC, USA, 2001. IEEE Computer Society.

[216] Y. Liu and T. M. Khoshgoftaar. Reducing overfitting in genetic programming models for software quality classification. In *Proceedings of the 8th IEEE International Symposium on High-Assurance Systems Engineering (HASE'04)*, Washington, DC, USA, 2004. IEEE Computer Society.

[217] Y. Liu, T. M. Khoshgoftaar, and N. Seliya. Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Transactions on Software Engineering*, 36(6):852–864, 2010.

[218] Y. Liu, T. M. Khoshgoftaar, and J.-F. Yao. Developing an effective validation strategy for genetic programming models based on multiple datasets. In *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration (IRI'06)*, 2006.

[219] J. Long. NASA IV&V metrics data program (MDP) data repository. `http://mdp.ivv.nasa.gov/`, 2011. Last checked: 15 March 2011.

[220] G. C. Low and D. R. Jeffery. Function points in the estimation and evaluation of the software process. *IEEE Transactions on Software Engineering*, 16(1):64–71, 1990.

[221] S. Luke and L. Panait. Lexicographic parsimony pressure. In *Proceedings of the 2002 Annual Genetic and Evolutionary Computation Conference (GECCO'02)*, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[222] S. Luke and L. Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, 2006.

[223] M. R. Lyu. *Handbook of software reliability engineering*. IEEE Computer Society Press and McGraw-Hill, 1996.

[224] M. R. Lyu. Software reliability engineering: A roadmap. In *Proceedings of Future of Software Engineering at 29th International Conference on Software Engineering (FOSE'07)*, Washington, DC, USA, 2007. IEEE Computer Society.

[225] M. R. Lyu and A. P. Nikora. Applying reliability models more effectively. *IEEE Software*, 9(4), 1992.

[226] Y. Ma and B. Cukic. Adequate and precise evaluation of quality models in software engineering studies. In *Proceedings of the 3rd International Workshop on Predictor Models in Software Engineering (PROMISE'07)*, Washington, DC, USA, 2007. IEEE Computer Society.

[227] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. Shepperd, and S. Webster. An investigation of machine learning based prediction systems. *Journal of Systems and Software*, 53(1):23–29, 2000.

[228] C. Mair and M. Shepperd. The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In *Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE'05)*, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[229] Y. K. Malaiya, N. Karunanithi, and P. Verma. Predictability measures for software reliability models. In *Proceedings of the 14th Annual International Computer Software and Applications Conference (COMPSAC'90)*, 1990.

[230] The MathWorks, Inc. `http://www.mathworks.com`. Last checked: 20 April 2008.

[231] K. Matsumoto, K. Inoue, T. Kikuno, and K. Torii. Experimental evaluation of software reliability growth models. In *Proceedings of the 18th International Symposium on Fault-Tolerant Computing (FTCS-18)*, 1988.

[232] T. J. McCabe. A complexity measure. In *Proceedings of the 2nd International Conference on Software Engineering (ICSE'76)*, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[233] P. McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2):105–156, 2004.

[234] T. Menzies, J. DiStefano, A. Orrego, and R. M. Chapman. Assessing predictors of software defects. In *Proceedings of the Workshop on Predictive Software Models, collocated with International Conference on Software Maintenance (ICSM'04)*, 2004. `http://menzies.us/pdf/04psm.pdf`.

[235] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[236] T. Menzies, O. Jalali, J. Hihn, D. Baker, and K. Lum. Stable rankings for different effort models. *Automated Software Engineering*, 17(4):409–437, 2010.

[237] Z. Michalewicz and D. B. Fogel. *How to solve it: Modern heuristics*. Springer Verlag, second edition, 2004.

[238] N. Mittas and L. Angelis. Comparing cost prediction models by resampling techniques. *Journal of Systems and Software*, 81(5):616–632, 2008.

[239] A. Mockus, D. M. Weiss, and P. Zhang. Understanding and predicting effort in software projects. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*. IEEE Computer Society, 2003.

[240] P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz. An empirical study of software reuse vs. defect-density and stability. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, Washington, DC, USA, 2004. IEEE Computer Society.

[241] L. C. Molina, L. Belanche, and À. Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, Washington, DC, USA, 2002. IEEE Computer Society.

[242] A. M. Molinaro, R. Simon, and R. M. Pfeiffer. Prediction error estimation: A comparison of resampling methods. *Bioinformatics*, 21(15):3301–3307, 2005.

[243] D. P. Muni, N. R. Pal, and J. Das. Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(1):106–117, 2006.

[244] J. Munson and T. M. Khoshgoftaar. Regression modelling of software quality: Empirical investigation. *Journal of Electronic Materials*, 19(6):106–114, 1990.

[245] J. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, 18(5):423–433, 1992.

[246] J. D. Musa. *Software reliability engineering: More reliable software faster and cheaper*. AuthorHouse, 2nd edition, 2004.

[247] G. J. Myers. *Art of software testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979.

[248] I. Myrtveit and E. Stensrud. A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE Transactions on Software Engineering*, 25(4), 1999.

[249] I. Myrtveit and E. Stensrud. A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE Transactions on Software Engineering*, 25(4):510–525, 1999.

[250] I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering*, 31(5):380–391, 2005.

[251] I. J. Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1), 2003.

[252] N. Nagappan and T. Ball. Static analysis tools as early indicators of pre-release defect density. In *Proceedings of the 27th International Conference on Software Engineering (ICSE'05)*, New York, NY, USA, 2005. ACM.

[253] N. Nagappan, B. Murphy, and V. R. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, New York, NY, USA, 2008. ACM.

[254] P. M. Nardi. *Doing survey research: A guide to quantitative methods*. Pearson Education, Inc., 2003.

[255] D. E. Neuman. An enhanced neural network technique for software risk analysis. *IEEE Transactions on Software Engineering*, 28(9):904–912, 2002.

[256] A. P. Nikora. CASRE homepage. `http://www.openchannelfoundation.org/projects/CASRE_3.0/`, 2008. Last checked: 20 April 2008.

[257] A. P. Nikora and M. R. Lyu. An experiment in determining software reliability model applicability. In *Proceedings of the 6th International Symposium on Software Reliability Engineering (ISSRE'95)*, 1995.

[258] M. Ohlsson and P. Runeson. Experience from replicating empirical studies on prediction models. In *Proceedings of the 8th IEEE Symposium on Software Metrics (METRICS'02)*, 2002.

[259] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, 22(12):886–894, 1996.

[260] N. Ohlsson, A. C. Eriksson, and M. Helander. Early risk-management by identification of fault-prone modules. *Empirical Software Engineering*, 2(2):166–173, 1997.

[261] N. Ohlsson, M. Zhao, and M. Helander. Application of multivariate analysis for software fault prediction. *Software Quality Journal*, 7(1):51–66, 1998.

[262] E. Oliveira, A. Pozo, and S. Vergilio. Using boosting techniques to improve software reliability models based on genetic programming. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, Washington, DC, USA, 2006. IEEE Computer Society.

[263] A. N. Oppenheim. *Questionnaire design, interviewing and attitude measurement*. Pinter Publishers, 1996.

[264] T. Ostrand and E. J. Weyuker. The distribution of faults in a large industrial software system. In *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'02)*, New York, NY, USA, 2002. ACM.

[265] T. Ostrand, E. J. Weyuker, and R. M. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[266] L. Ozdamar and G. Ulusoy. A survey on the resource-constrained project scheduling problem. *IIE Transactions*, 27:574–586, 1995.

[267] L. Pelayo and S. Dick. Applying novel resampling strategies to software defect prediction. In *Annual meeting of the North American Fuzzy Information Processing Society (NAFIPS'07)*, 2007.

[268] M. Petticrew and H. Roberts. *Systematic reviews in the social sciences: A practical guide*. Wiley-Blackwell, Victoria, Australia, 2005.

[269] H. Pham. *Software reliability*. Springer-Verlag, Singapore, 2000.

[270] L. M. Pickard, B. A. Kitchenham, and S. J. Linkman. An investigation of analysis techniques for software datasets. In *Proceedings of the 6th International Software Metrics Symposium (METRICS'99)*, Los Alamitos, USA, 1999. IEEE Computer Society.

[271] D. G. Pitt and D. P. Kreutzweiser. Applications of computer-intensive statistical methods to environmental research. *Ecotoxicology and Environmental Safety*, 39(2):78 – 97, 1998.

[272] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[273] R. S. Pressman. *Software engineering – A practitioner's approach.* McGraw-Hill Higher Education, fifth edition, 2001.

[274] L. H. Putnam. A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, 4(4):345–361, 1978.

[275] S. R. Rakitin. *Software verification and validation for practitioners and managers.* Artech House., Inc., 685 Canton Street, Norwood, MA, USA, second edition, 2001.

[276] R. B. Rao and G. Fung. On the dangers of cross-validation. An experimental evaluation. In *Proceedings of the SIAM International Conference on Data Mining (SDM'08)*, 2008.

[277] J. Ratzinger, H. Gall, and M. Pinzger. Quality assessment based on attribute series of software evolution. In *Proceedings of the 14th Working Conference on Reverse Engineering (WCRE'07)*, 2007.

[278] M. Reformat, W. Pedrycz, and N. J. Pizzi. Software quality analysis with the use of computational intelligence. *Information and Software Technology*, 45(7):405 – 417, 2003.

[279] E. N. Regolin, G. A. de Souza, A. Pozo, and S. Vergilio. Exploring machine learning techniques for software size estimation. In *Proceedings of the International Conference of the Chilean Computer Science Society (SCCC'03)*, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[280] G. Robinson and P. McIlroy. Exploring some commercial applications of genetic programming. In *Selected Papers from AISB Workshop on Evolutionary Computing*, London, UK, 1995. Springer–Verlag.

[281] C. Robson. *Real world research.* Blackwell Publishing, United Kingdom, second edition, 2002.

[282] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, 2006.

[283] B. J. Ross. The effects of randomly sampled training data on program evolution. In *Proceedings of the 2000 Annual Genetic and Evolutionary Computation Conference (GECCO'00)*, USA, 2000. Morgan Kaufmann.

[284] L. M. Rudner and M. M. Shafer. Resampling: A marriage of computers and statistics. *Practical Assessment, Research & Evaluation*, 3(5), 1992.

[285] P. Runeson, C. Andersson, T. Thelin, A. Andrews, and T. Berling. What do we know about defect detection methods? *IEEE Software*, 23(3):82–90, 2006.

[286] S. Russell and P. Norvig. *Artificial intelligence—A modern approach.* Prentice-Hall Series in Artificial Intelligence, USA, 2003.

[287] C. Schaffer. Selecting a classification method by cross-validation. *Machine Learning*, 13(1):135–143, 1993.

[288] Y. Shan, R. I. McKay, C. J. Lokan, and D. L. Essam. Software project effort estimation using genetic programming. In *Proceedings of the 2002 International Conference on Communications, Circuits and Systems (ICCCAS'02)*, Piscataway, NJ, USA, 2002.

[289] M. Shepperd, M. Cartwright, and G. Kadoda. On building prediction systems for software engineers. *Empirical Software Engineering*, 5(3):175–182, 2000.

[290] M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, 2001.

[291] K. K. Shukla. Neuro-genetic prediction of software development effort. *Information and Software Technology*, 42(10):701–713, 2000.

[292] S. Silva. GPLAB—A genetic programming toolbox for MATLAB. `http://gplab.sourceforge.net`. Last checked: 30 March 2009.

[293] R. Sitte. Comparison of software reliability growth predictions: Neural networks vs. parametric recalibration. *IEEE Transactions on Reliability*, 48(3):285–291, 1999.

[294] M. G. Smith and L. Bull. Feature construction and selection using genetic programming and a genetic algorithm. In *Proceedings of the 6th European Conference on Genetic programming (EuroGP'03)*, Berlin, Heidelberg, 2003. Springer-Verlag.

[295] S. F. Smith. *A learning system based on genetic adaptive algorithms.* PhD thesis, University of Pittsburgh, Pittsburgh, PA, USA, 1980.

[296] T. A. B. Snijders. On cross-validation for predictor evaluation in time series. *Dijkstra*, pages 56–69, 1988.

[297] S. S. So, S. D. Cha, and Y. R. Kwon. Empirical evaluation of a fuzzy logic-based software quality prediction model. *Fuzzy Sets and Systems*, 127(2):199–208, 2002.

[298] Software and systems engineering standards committee of the IEEE computer society, The Institute of Electrical and Electronic Engineers, Inc. New York, USA. *IEEE standard 12207-2008 systems and software engineering – Software life cycle processes*, 2008.

[299] Software Engineering Standards Coordinating Committee of the IEEE Computer Society, IEEE Standards Board, The Institute of Electrical and Electronic Engineers, Inc. New York, USA. *IEEE guide for software verification and validation plans – IEEE std 1059-1993*, 1993.

[300] Q. Song, M. Shepperd, M. Cartwright, and C. Mair. Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, 32(2), 2006.

[301] Standards Coordinating Committee of the Computer Society of the IEEE, IEEE Standards Board, The Institute of Electrical and Electronic Engineers, Inc. New York, USA. *IEEE standard glossary of software engineering terminology – IEEE std 610.12-1990*, 1990.

[302] Standards coordinating committee of the computer society of the IEEE, IEEE standards board, The institute of electrical and electronic engineers, inc. New York, USA. *IEEE standard for a software quality metrics methodology – IEEE std 1061-1998*, 1998.

[303] M. Staron and W. Meding. Predicting weekly defect inflow in large software projects based on project planning and test status. *Information and Software Technology*, 50(7–8):782–796, 2008.

[304] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, Series B-36:111–147, 1974.

[305] C. Stringfellow and A. A. Andrews. An empirical method for selecting software reliability growth models. *Empirical Software Engineering*, 7(4):319–343, 2002.

[306] Y. Su and C. Huang. Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. *Journal of Systems and Software*, 80(4):606–615, 2007.

[307] T. Thelin. Team-based fault content estimation in the software inspection process. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, 2004.

[308] B. Thompson and P. A. Snyder. Statistical significance testing practices. *The Journal of Experimental Education*, 66(1):75–83, 1997.

[309] J. Tian. An integrated approach to test tracking and analysis. *Journal of Systems and Software*, 35(2):127–140, 1996.

[310] J. Tian. Quality-evaluation models and measurements. *IEEE Software*, 21(3):84–91, 2004.

[311] L. Tian and A. Noore. Software reliability prediction by soft computing techniques. *International Journal of Neural Systems*, 14(3):165–174, 2004.

[312] L. Tian and A. Noore. Dynamic software reliability prediction: An approach based on support vector machines. *International Journal of Reliability, Quality and Safety Engineering*, 12(4):309–321, 2005.

[313] L. Tian and A. Noore. Evolutionary neural network modeling for software cumulative failure time prediction. *International Journal of Reliability, Quality and Safety Engineering*, 87(1):45–51, 2005.

[314] L. Tian and A. Noore. On-line prediction of software reliability using an evolutionary connectionist model. *Journal of Systems and Software*, 77(2):173–180, 2005.

[315] L. Tian and A. Noore. *Computational intelligence in reliability engineering*, chapter Computational intelligence methods in software reliability prediction, pages 375–398. Springer Berlin / Heidelberg, 2007.

[316] L. Tian and A. Noore. Computational intelligence methods in software reliability prediction. *Computational Intelligence in Reliability Engineering*, 39:375–398, 2007.

[317] P. Tomaszewski, J. Håkansson, H. Grahn, and L. Lundberg. Statistical models vs. expert estimation for fault prediction in modified code – An industrial case study. *Journal of Systems and Software*, 80(8):1227–1238, 2007.

[318] C. Torgerson. *Systematic reviews*. Continuum International Publishing Group, 2003.

[319] A. Tosun and A. Basar. AR1 data set. `http://promisedata.org/repository/data/ar/ar1.arff`, 2011. Last checked: 15 March 2011.

[320] A. Tosun and A. Basar. AR6 data set. `http://promisedata.org/repository/data/ar/ar6.arff`, 2011. Last checked: 15 March 2011.

[321] I. C. Trelea. The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters*, 85(6), 2003.

[322] A. Tsakonas and G. Dounias. Predicting defects in software using grammar-guided genetic programming. In *Proceedings of the 5th Hellenic conference on Artificial Intelligence (SETN'08)*, Berlin, Heidelberg, 2008. Springer-Verlag.

[323] L. Utkin, S. Gurov, and M. Shubinsky. A fuzzy software reliability model with multiple-error introduction and removal. *International Journal of Reliability, Quality and Safety Engineering*, 9(3):215–227, 2002.

[324] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. De Backer, and R. Haesen. Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and software*, 81(5):823–839, 2008.

[325] A. Veevers and A. C. Marshall. A relationship between software coverage metrics and reliability. *Software Testing, Verification and Reliability*, 4(1):3–8, 1994.

[326] E. Vladislavleva. *Model-based problem solving through symbolic regression via Pareto genetic programming*. PhD thesis, Tilburg university, Holland, 2008.

[327] D. Vose. *Risk analysis: A quantitative guide*. John Wiley & Sons, Ltd., England, second edition, 2001.

[328] Ž. Antolić. Fault slip through measurement process implementation in CPP software implementation. In *Proceedings of the 30th Jubilee International Convention (MIPRO'07), Ericsson Nikola Tesla*, 2007.

[329] S. Wagner. A literature survey of the quality economics of defect-detection techniques. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering (ISESE'06)*, 2006.

[330] M. B. Wall. *A genetic algorithm for resource-constrained scheduling*. PhD thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, USA, 1996.

[331] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye. *Probability and statistics for engineers and scientists*. Prentice-Hall, New Jersey, USA, 2002.

[332] H. Wang, T. M. Khoshgoftaar, K. Gao, and N. Seliya. High-dimensional software engineering data and feature selection. In *21st International Conference on Tools with Artificial Intelligence (ICTAI'09)*, 2009.

[333] M. C. Wang. Re-sampling procedures for reducing bias of error rate estimation in multinomial classification. *Computational Statistics and Data Analysis*, 4(1):15–39, 1986.

[334] Y. Wang. *A new approach to fitting linear models in high dimensional spaces*. PhD thesis, Department of Computer Science, University of Waikato, New Zealand, 2000.

[335] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *Proceedings of the 9th European Conference on Machine Learning (Poster papers) (ECML'95)*, 1995.

[336] A. Watkins, J. Timmis, and L. Boggess. Artificial immune recognition system (AIRS): An immune-inspired supervised learning algorithm. *Genetic programming and Evolvable Machines*, 5(3), 2004.

[337] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07)*. IEEE, 2007.

[338] Weka documentation. Class reptree.tree. `http://www.dbs.ifi.lmu.de/~zimek/diplomathesis/implementations/EHNDs/doc/weka/classifiers/trees/REPTree.Tree.html`, 2011. Last checked: 15 March 2011.

[339] E. J. Weyuker, T. Ostrand, and R. M. Bell. Comparing the effectiveness of several modeling methods for fault prediction. *Empirical Software Engineering*, 15(3):277–295, 2010.

[340] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[341] I. H. Witten and E. Frank. *Data mining—Practical machine learning tools and techniques*. Morgan Kaufmann Publishers, USA, 2005.

[342] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: An introduction*. Kluwer Academic Publishers, USA, 2000.

[343] A. Wood. Predicting software reliability. *Computer*, 29(11), 1996.

[344] A. Wood. Software reliability growth models: Assumptions vs. reality. In *Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering (ISSRE'97)*, Los Alamitos, CA, USA, 1997. IEEE Computer Society.

[345] J. Xiao, Q. Wang, M. Li, Q. Yang, L. Xie, and D. Liu. Value-based multiple software projects scheduling with genetic algorithm. In *Proceedings of the 2009 International Conference on Software Process (ICSP'09)*, 2009.

[346] J. Xiao, Q. Wang, M. Li, Y. Yang, F. Zhang, and L. Xie. A constraint-driven human resource scheduling method in software development and maintenance process. In *24th IEEE International Conference on Software Maintenance (ICSM'08)*. IEEE, 2008.

[347] S. Yamada, M. Ohba, and S. Osaki. S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability*, R-32(5):475 – 478, 1983.

[348] J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems and their Applications*, 13(2):44–49, 1998.

[349] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the International Conference on Machine Learning (ICML'97)*, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[350] W. A. Yousef, R. F. Wagner, and M. H. Loew. Comparison of non-parametric methods for assessing classifier performance in terms of ROC parameters. In *Proceedings of the 33rd Applied Imagery Pattern Recognition Workshop (AIPR'04)*, Washington, DC, USA, 2004. IEEE Computer Society.

[351] C. H. Yu. Resampling methods: Concepts, applications, and justification. *Practical Assessment, Research & Evaluation*, 8(19), 2003.

[352] T. J. Yu, V. Y. Shen, and H. E. Dunsmore. An analysis of several software defect models. *IEEE Transactions on Software Engineering*, 14(9):1261–1270, 1988.

[353] H. Zeng and D. Rine. Estimation of software defects fix effort using neural networks. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04) – Workshops and Fast Abstracts*. IEEE Computer Society, 2004.

[354] D. Zhang and J. J. P. Tsai. Machine learning and software engineering. *Software Quality Control*, 11(2):87–119, 2003.

[355] Y. Zhang and H. Chen. Predicting for MTBF failure data series of software reliability by genetic programming algorithm. In *Proceedings of the 6th International Conference on Intelligent Systems Design and Applications (ISDA'06)*, Washington, DC, USA, 2006. IEEE Computer Society.

[356] Y. Zhang and J. Yin. Software reliability model by AGP. In *Proceedings of the IEEE International Conference on Industrial Technology (ICIT'08)*, Piscataway, NJ, United States, 2008.

[357] S. Zhong, T. M. Khoshgoftaar, and N. Seliya. Unsupervised learning for expert-based software quality estimation. In *Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering (HASE'04)*, 2004.

Appendix A: Study quality assessment (Chapter 2)

Table 1: Study quality assessment.

Criteria
A: Are the aims of the research/research questions clearly stated? *B*: Do the study measures allow the research questions to be answered? *C*: Is the sample representative of the population to which the results will generalize? *D*: Is there a comparison group? *E*: Is there an adequate description of the data collection methods? *F*: Is there a description of the method used to analyze data? *G*: Was statistical hypothesis testing undertaken? *H*: Are all study questions answered? *I*: Are the findings clearly stated and relate to the aims of research? *J*: Are the parameter settings for the algorithms given? *K*: Is there a description of the training and testing sets used for the model construction methods?

(a) Study quality assessment criteria

	[280]	[96]	[180]	[215]	[173]	[174]	[216]	[278]	[218]
A	√	√	√	√	√	√	√	√	√
B	√	√	√	√	√	√	√	√	√
C	~×	~√	~√	~√	~√	~√	~√	~√	~√
D	√	√	√	√	√	√	√	√	√
E	~√	~√	~√	~√	~√	~√	~√	~√	~√
F	√	√	√	√	√	√	√	√	√
G	×	×	×	×	×	×	×	×	×
H	√	√	√	√	√	√	√	√	√
I	√	√	√	√	√	√	√	√	√
J	~√	√	~√	√	√	√	~√	×	√
K	√	√	√	√	√	√	√	√	√

(b) Study quality assessment for software quality classification studies

	[89]	[87]	[279]	[88]	[56]	[288]	[208]
A	√	√	√	√	√	√	√
B	√	√	√	√	√	√	√
C	√	~√	√	√	√	√	√
D	√	√	√	√	√	√	√
E	~√	√	√	√	√	√	√
F	√	√	√	√	√	√	√
G	×	×	×	×	~√	×	×
H	√	√	√	√	√	√	√
I	√	√	√	√	√	√	√
J	√	√	√	√	√	√	√
K	√	√	√	√	√	√	√

(c) Study quality assessment for software CES estimation

	[160]	[161]	[322]	[355]	[356]	[6]	[73]	[262]
A	√	√	√	√	√	√	√	√
B	√	√	√	√	√	√	√	√
C	~√	~√	~√	~√	~√	√	√	~√
D	√	√	√	√	√	√	√	√
E	√	√	√	×	×	~√	~√	√
F	√	√	√	~√	~√	√	√	√
G	√	√	×	×	×	√	√	√
H	√	√	√	√	√	√	√	√
I	√	√	√	~√	~√	√	√	√
J	~√	~√	√	~√	~√	√	√	√
K	×	×	√	×	×	√	√	√

(d) Study quality assessment for software fault prediction and software reliability growth modeling

Appendix B: Model training procedure (Chapter 5)

This appendix discusses the parameter settings that have been considered for different techniques during model selection. These settings may be used for a future replication of this study and to quantify the impact of changing the parameter settings, perhaps using different data sets. As given in Section 5.3.1, we use data from 45 weeks of the baseline project to train the models while the results are evaluated on the data from 15 weeks of an on-going project. The experimental evaluation process is also summarized in Procedure 1.

The least-square multiple regression does not require selection of parameters, rather the coefficients are determined from the training data. Different estimators implemented in the WEKA machine learning tool [123] have been evaluated for pace regression, that includes empirical Bayes, ordinary least square, Akaike's information criterion (AIC) and risk inflation criterion (RIC). The estimator giving the least ARE is selected as the best pace regression model.

The M5P technique requires setting the minimum number of instances at a leaf node and has been varied in the range [2, 4, ..., 10] with pruning and smoothing. The model with minimum ARE is retained. The REPTree technique requires setting the maximum depth of the tree, the minimum total weight of the instances in a leaf, the minimum variance proportion at a node required for splitting, the number of folds of data used for pruning and the seed value used for randomizing the data. We have imposed no restriction on the maximum depth of the tree while the minimum total weight of the instances in a leaf is varied in the range [2,4, ..., 10]. The minimum variance proportion at a node, the number of folds of data used for pruning and the seed value used for randomization are kept constant at their default values of 0.0010, 3 and 1 respectively.

Procedure 1 The model training procedure.

```
Train: Training dataset of 45 weeks
Test: Testing dataset of 15 weeks
P: Set of parameter settings for each technique
T: Set of techniques
for each t in T do
    for each p in P do
        Model = BuildModel (Train, t, P[t])
        ARE [t, p] = GetResult (Test, Model)
        output ARE
    end for
    return parameter[min(ARE)]
end for
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
BuildModel (TrainingData, Technique, ParameterSet)
// Train the technique using training data with parameters
GetResult (TestData, Model)
// Compute ARE of the trained model on the Test data
```

The K* instance-based technique requires setting the blending parameter that has a value between 0% and 100%. This parameter has been varied in the range of [0, 20, 40, . . . , 100]. For k-NN, the number of neighbors has been varied in the range of [1, 3, 5, . . . , 15].

For SVM, two types of parameters have to be set by the user, i.e., values for the epsilon parameter, ε and the regularization parameter, C. Setting the value of C near the range of the output values has been found to be a successful heuristic. We therefore vary C within the range [1, 3, . . . , 11]. The value of ε is varied in the range [0.001, 0.003] while the kernel used is the radial basis function. Training an artificial neural network (ANN) requires deciding on the number of layers and the number of nodes at each layer. We considered the ANN architecture with 1 input layer, 2 hidden layers and 1 output layer. The number of independent variables in the problem determined the number of input nodes. The two hidden layers used a varied number of nodes in the range [1, 3, 5, 7], while the output layer used a single node. The hyperbolic tangent sigmoid and linear transfer functions have been used for the hidden and output nodes respectively. Finally the number of epochs used is 500 and the weights are updated using a learning rate of 0.3 and a momentum of 0.2.

Model selection for Bagging involves deciding upon the size of the bag as a per-

Table 2: GP control parameters.

Control parameter	Value
Population size	50
Termination condition	2000 generations
Function set	$\{+,-,*,/,\sin,\cos,\log,\text{sqrt}\}$
Tree initialization	Ramped half-and-half method
Probabilities of crossover, mutation, reproduction	0.8, 0.1, 0.1
Selection method	roulette-wheel

centage of the training set size and the number of iterations to be performed. These two parameters have been varied in the range [25, 50, 75, 100] and [5, 10, 15] respectively. The REPTree technique is used as the base learner. For rotation forest, the number of iterations have been varied in the range [5, 10, 15] and the base learner used is the REPTree technique.

GP requires setting a number of control parameters. Although the affect of changing these control parameters on the end solution is still an active area of research, we nevertheless experimented with different function and terminal sets. Initially we experimented with a minimal set of functions and the terminal set containing the independent variable only. We incrementally increased the function set with additional functions and later on also complemented the terminal set with a random constant. The best model having the best fitness was chosen from all the runs of the GP system with different variations of function and terminal sets. The GP programs were evaluated according to the sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$, where e_i is the actual fault count data, $e_i^{'}$ is the estimated value of the fault count data and n is the size of the data set used to train the GP models. The control parameters that were chosen for the GP system are shown in Table 2.

For GEP, the solutions are evaluated for fitness using mean squared error and the control parameters are shown in Table 3.

The AIRS algorithm also require setting a number of parameters. While it is not possible to experiment with all the different combinations of these parameters, however the value of k for the majority voting has been varied in the range [1, 3, 5, . . . , 15]. Rest of the parameters used were: Affinity threshold = 0.2, clonal rate = 10, hypermutation rate = 2, mutation rate = 0.1, stimulation value = 0.9 and total resources = 150. For PSO-ANN, the architecture similar to the basic ANN is followed except that the weights are now optimized using PSO with the number of particles in the swarm set to 25 and the number of iterations varied in the range [500, 1000, 15000, 2000]. The

Table 3: GEP control parameters.

Control Parameter	Value
Population size	50
Genes per chromosome	4
Gene head length	8
Termination condition	2000 generations
Functions	$\{+,-,*,/,$sin,cos,log,sqrt$\}$
Tree initialization	Random
Mutation rate, Inversion rate, IS transposition rate, Root transposition rate	0.04, 0.1, 0.1, 0.1
Gene transposition rate, One-point recombination rate	0.1, 0.3
Two-point recombination rate, Gene recombination rate	0.3, 0.1
Selection method	roulette-wheel

mean squared error is used as the fitness function.

LIST OF FIGURES

LIST OF TABLES

In preparing for battle, I have always found that plans are useless,
but planning is indispensable.
Dwight D. Eisenhower (1890–1969)

ABSTRACT

Software verification and validation (V&V) activities are critical for achieving software quality; however, these activities also constitute a large part of the costs when developing software. Therefore efficient and effective software V&V activities are both a priority and a necessity considering the pressure to decrease time-to-market and the intense competition faced by many, if not all, companies today. It is then perhaps not unexpected that decisions that affects software quality, e.g., how to allocate testing resources, develop testing schedules and to decide when to stop testing, needs to be as stable and accurate as possible.

The objective of this thesis is to investigate how search-based techniques can support decision-making and help control variation in software V&V activities, thereby indirectly improving software quality. Several themes in providing this support are investigated: predicting reliability of future software versions based on fault history; fault prediction to improve test phase efficiency; assignment of resources to fixing faults; and distinguishing fault-prone software modules from non-faulty ones. A common element in these investigations is the use of search-based techniques, often also called metaheuristic techniques, for supporting the V&V decision-making processes. Search-based techniques are promising since, as many problems in real world, software V&V can be formulated as optimization problems where near optimal solutions are often good enough. Moreover, these techniques are general optimization solutions that can potentially be applied across a larger variety of decision-making situations than other existing alternatives. Apart from presenting the current state of the art, in the form of a systematic literature review, and doing comparative evaluations of a variety of metaheuristic techniques on large-scale projects (both industrial and open-source), this thesis also presents methodological investigations using search-based techniques that are relevant to the task of software quality measurement and prediction.

The results of applying search-based techniques in large-scale projects, while investigating a variety of research themes, show that they consistently give competitive results in comparison with existing techniques. Based on the research findings, we conclude that search-based techniques are viable techniques to use in supporting the decision-making processes within software V&V activities. The accuracy and consistency of these techniques make them important tools when developing future decision-support for effective management of software V&V activities.

BLEKINGE TEKNISKA HÖGSKOLA · BTH ·

ISSN 1653-2090
ISBN 978-91-7295-203-4

2011:06