# SEARCH-BASED APPROACHES TO SOFTWARE FAULT PREDICTION AND SOFTWARE TESTING

Wasif Afzal

Blekinge Institute of Technology
Licentiate Dissertation Series No. 2009:06

School of Engineering

# Search-Based Approaches to Software Fault Prediction and Software Testing

Wasif Afzal

Blekinge Institute of Technology Licentiate Dissertation Series
No 2009:06

# Search-Based Approaches to Software Fault Prediction and Software Testing

**Wasif Afzal**

Department of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
SWEDEN

© 2009 Wasif Afzal
Department of Systems and Software Engineering
School of Engineering
Publisher: Blekinge Institute of Technology
Printed by Printfabriken, Karlskrona, Sweden 2009
ISBN 978-91-7295-163-1
Blekinge Institute of Technology Licentiate Dissertation Series
ISSN 1650-2140
urn:nbn:se:bth-00439

*In the valley of the blind, the one-eyed man is king.*
Gerard Erasmus (circa 1500)

This thesis is submitted to the Research Board at Blekinge Institute of Technology, in partial fulfillment of the requirements for the degree of Licentiate of Engineering.

**Contact Information:**

Wasif Afzal
Blekinge Institute of Technology
P.O. Box 520
SE-372 25 Ronneby
SWEDEN

Tel: +46 457 385 840
Fax: +46 457 279 14
E-mail: wasif.afzal@bth.se

# Abstract

Software verification and validation activities are essential for software quality but also constitute a large part of software development costs. Therefore efficient and cost-effective software verification and validation activities are both a priority and a necessity considering the pressure to decrease time-to-market and intense competition faced by many, if not all, companies today. It is then perhaps not unexpected that decisions related to software quality, when to stop testing, testing schedule and testing resource allocation needs to be as accurate as possible.

This thesis investigates the application of search-based techniques within two activities of software verification and validation: Software fault prediction and software testing for non-functional system properties. Software fault prediction modeling can provide support for making important decisions as outlined above. In this thesis we empirically evaluate symbolic regression using genetic programming (a search-based technique) as a potential method for software fault predictions. Using data sets from both industrial and open-source software, the strengths and weaknesses of applying symbolic regression in genetic programming are evaluated against competitive techniques. In addition to software fault prediction this thesis also consolidates available research into predictive modeling of other attributes by applying symbolic regression in genetic programming, thus presenting a broader perspective. As an extension to the application of search-based techniques within software verification and validation this thesis further investigates the extent of application of search-based techniques for testing non-functional system properties.

Based on the research findings in this thesis it can be concluded that applying symbolic regression in genetic programming may be a viable technique for software fault prediction. We additionally seek literature evidence where other search-based techniques are applied for testing of non-functional system properties, hence contributing towards the growing application of search-based techniques in diverse activities within software verification and validation.

# Acknowledgements

First of all, I am grateful to my advisors, Dr. Richard Torkar and Dr. Robert Feldt, for their invaluable support throughout the research. Their guidance and ideas have been instrumental in steering this research; this thesis would not have been a reality without their continued support and encouragement. I am still to learn a lot from them. Secondly, I am thankful to Prof. Claes Wohlin for allowing me the opportunity to undertake post-graduate studies and to be part of the SERL research group. I am also thankful to him for reading the thesis chapters and providing crucial feedback.

I am also thankful to our industrial contacts who have been responsive to requests for data sets. It would not have been possible to complete this thesis without their help. 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. I thank them for offering me learning opportunities through interaction and course work.

It will be unjust not to mention the support I got from the library staff at Infocenter in Ronneby, especially Kent Pettersson and Eva Norling helped me find research papers and books on several occasions. Kent Adolfsson assisted me in the economical matters while May-Louise Andersson, Eleonore Lundberg and Monica H. Nilsson provided me 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. Lastly, I am grateful to my father who passed away in 2003 but not before he had influenced my personality to be what I am today.

The research in this thesis was funded by Sparbanksstiftelsen Kronan and the Knowledge Foundation in Sweden under a research grant for the project "Blekinge – Engineering Software Qualities (BESQ)".

# Overview of Papers

Chapter 2 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).

Chapters 3 and 4 have been submitted to the *Journal of Empirical Software Engineering* and the *Journal of Systems and Software* respectively.

Chapter 5 is an extended version of the paper "A systematic mapping study on non-functional search-based software testing" – published in the proceedings of the *20th International Conference on Software Engineering and Knowledge Engineering* (SEKE'08). The extended version, entitled "A systematic review of search-based testing for non-functional system properties" was published in the *Journal of Information and Software Technology*.

Wasif Afzal is the main author and Dr. Richard Torkar is the co-author of all the chapters in this thesis. Dr. Robert Feldt is the co-author of Chapters 2, 3 and 5; Dr. Tony Gorscheck is the additional co-author of Chapter 3.

Papers that were related but not included in the thesis:

(i) W. Afzal, 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).

(ii) W. Afzal, 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).

(iii) R. Feldt, R. Torkar, T. Gorschek, 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).

# Contents

# Chapter 1

# Introduction

## 1.1 Preamble

The IEEE Standard Glossary of Software Engineering Terminology [232] 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. During the course of a software development life cycle, certain surrounding activities [210] occur, and software verification and validation (V&V) is one example of such an activity. The collection of software V&V activities is also often termed as Software Quality Assurance (SQA) activities.

Software V&V consists of two distinct set of activities. Verification consists of a set of activities that checks the correct implementation of a specific function; while validation is a name given to a set of activities that checks that the software satisfies customer requirements. The IEEE Guide for Software Verification and Validation Plans [231] 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 [28] 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?"

Although one normally do not make a clear-cut distinction between software V&V activities, because a degree of overlap is inevitable, we conform to the software V&V

activities as given by Rakitin [213] for the purpose of brevity. According to Rakitin [213], while measurement is common to both V&V activities, verification activities additionally include inspection and configuration management; while validation activities additionally encompass testing and software reliability growth. Figure 1.1 presents a holistic view of software V&V activities as these are represented as occurring throughout the software development life cycle.

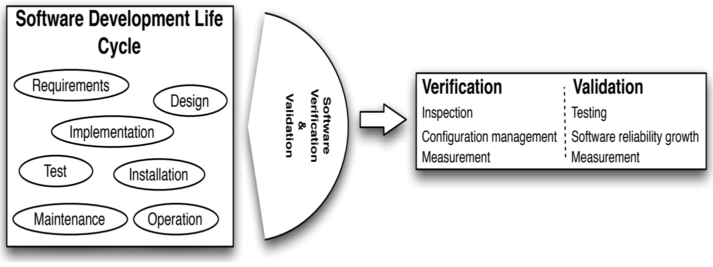


Figure 1.1: The software V&V activities occurs through out a software development life cycle.

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 [111]).

The overarching purpose of software V&V activities is to improve software product quality. While literature offers different definitions of the term 'software product quality', one common and decisive element in determining software product quality is adherence to user/customer requirements. These requirements are the properties that must be exhibited by the software to solve some real-world problem [111]. The requirements can further be classified into functional and non-functional (quality) requirements. While functional requirements are concerned with the functionality/capability of a software; non-functional requirements act as constraints to the solution that defines the desired quality attributes. Therefore, software product quality refers to the conformance of both functional and non-functional requirements.

It is quite obvious to reason that efficient and cost-effective software V&V activities increase our chances of delivering quality software to the 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 total software development life cycle costs. According to Boehm and Basili, around 40% [27], while Myers [189] argues that detection and removal of faults constitutes around 50% of project budgets.

We live today in a competitive global economy where time-to-market is of utmost importance [213]. At the same time, 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. We argue that trend analysis based on the number of faults found during software testing is one step towards this goal.

A major part of this thesis investigates the possibility of analyzing software fault history as a measurement technique to predict future software reliability. 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.

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 to fault prediction modeling is more recent [265]. 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 [18]. Within software engineering predictive modeling, machine learning has been applied for the tasks of classification and regression [265]. 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[1], complex non-linear relationships, outliers and small size of the data sets [90]. Various machine learning algorithms have been applied for software fault prediction; a non-exhaustive summary is provided in Section 3.2 of this thesis.

Our focus in this thesis is to make use of evolutionary computation approaches to machine learning called evolutionary learning (EL) [262]. Evolutionary computation is a collection of population-based algorithms making use of simulated evolution, random variation and selection [21]. One major branch of evolutionary computation

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

is genetic programming [157]. We have performed empirical studies making use of genetic programming (GP) to predict future software reliability in terms of fault count.

The reasons for carrying out these type of studies were:

1. To make use of advantages offered by genetic programming as a potential prediction tool:
    (a) GP models do not depend on assumptions about data distribution and relationship between independent and dependent variables.
    (b) GP models are independent of any assumptions about the stochastic behavior of the software failure process and the nature of software faults.
    (c) GP models do not conceive a particular structure for the resulting model.
    (d) The model and the associated coefficients can be evolved based on the historical fault data.
2. To evaluate earlier published results using genetic programming for fault predictions.
3. To investigate the application of GP models in the current trend of multi-release software development projects.
4. To consolidate the existing evidence in support (or against the use) of GP as a prediction tool.

The above focus on evolutionary computation also grows out of an increasing interest in an emerging field within software engineering called search-based software engineering (SBSE) [105, 103]. Search-based software engineering concerns solving software engineering problems using search-based optimization techniques. GP is one example of search-based optimization techniques. Within SBSE, a wide-range of studies are focussed on the problem of automated software test data generation. A survey paper by McMinn [182] reviews the field of search-based software test data generation. This survey paper, in addition to other types of testing, highlighted the use of search-based techniques for testing the non-functional property of execution time. McMinn also suggested possible directions of future research into *other* non-functional search-based software testing areas. We, in this thesis, investigated exactly this, in the form of a systematic review [150], allowing us to answer one of the research questions in this thesis (Section 1.3). As discussed above that software product quality is conformance to both functional and non-functional requirements, therefore, testing for non-functional system properties cannot be neglected in the goal towards a quality software product. While there are a plethora of techniques for testing functional requirements, testing non-functional requirements is usually not as straightforward.

Figure 1.2 presents a snapshot of major concerns addressed in this thesis.

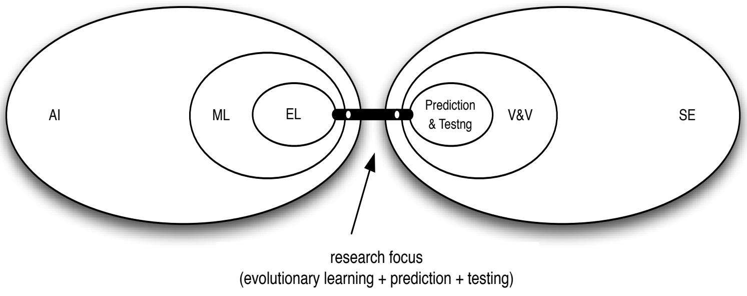


Figure 1.2: Cross-connecting concerns addressed in this thesis.

## 1.2 Concepts and Related work

It is clear that software engineering data, like any other data, becomes useful only when it is turned into information through analysis. This information can be used to make predictions; 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 a typical software engineering data is such that different machine learning techniques [46, 18] might be conducive to play a part in understanding a rather complex and changing software engineering process.

The Section 1.2 describes the concepts and their use in the thesis. We discuss the concepts of search-based software engineering, software fault prediction and systematic literature reviews.

### 1.2.1 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 [105, 102, 103]. 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 [46]. 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 [102]. Thus the human effort is focussed on guiding the automated search, rather than actually performing the search [102]. Certain problem characteristics warrant the application of search-techniques, which includes large number of possible solutions (search space) and no known optimal solutions [104]. Other desirable problem characteristics amenable to search-techniques' application include low computational complexity of fitness evaluations of potential solutions and continuity of the fitness function [104].

There are numerous examples of the applications of SBSE spanning over the whole software development life cycle, e.g. requirements engineering [22], project planning [11], software testing [182], software maintenance [29] and quality assessment [30].

### 1.2.2 Search-based software engineering (SBSE) in this thesis

This thesis addresses research questions that are related to SBSE and have a focus on: i) A particular problem domain and ii) Application of a specific or different search technique(s) on that particular domain.

A main part of this thesis includes software engineering predictive modeling as a problem domain while the technique used in this case is genetic programming. On the other hand, another problem domain in this thesis is software testing for non-functional system properties with the scope being broad to cater for different search techniques. The following Subsection presents an introduction to genetic programming.

#### Genetic programming

Genetic programming (GP) [157, 229] is an evolutionary computation technique and is an extension of genetic algorithms [108]. Like other evolutionary methods, GP is inspired by evolution in nature. It genetically breeds a population of computer programs [46] in pursuit of solving a problem. An abstract level definition of GP is given in [208] and reads as follows: "[GP] 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." GP applies iterative, random variation to an existing pool of computer programs (potential solutions to the problem) to form a new generation of programs by applying analogs of naturally occurring genetic operations [158]. The typical process, as given in [158], is depicted in Figure 1.3.

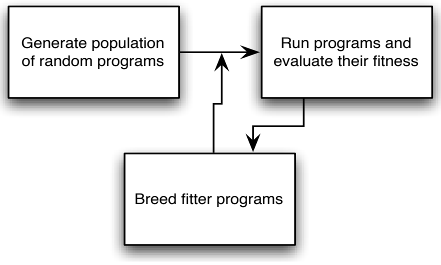

Generate population of random programs
Run programs and evaluate their fitness
Breed fitter programs

Figure 1.3: A block-diagram depicting GP evolutionary process.

Programs may be expressed in GP as syntax trees, with the nodes indicating the instructions to execute (called functions), while the tree leaves are called terminals and may consist of independent variables of the problem and random constants. In Figure 1.4, variables $x$, $y$ and constant 3 are the terminals while *min*, $*$, $+$ and $/$ are the functions.

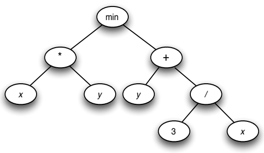


Figure 1.4: Tree structured representation showing $min(x*y, y+3/x)$.

To use GP one usually needs to take five preparatory steps [46]:

1. Specifying the set of terminals.
2. Specifying the set of functions.
3. Specifying the fitness measure.
4. Specifying the parameters for controlling the run.
5. Specifying the termination criterion and designating the result of the run.

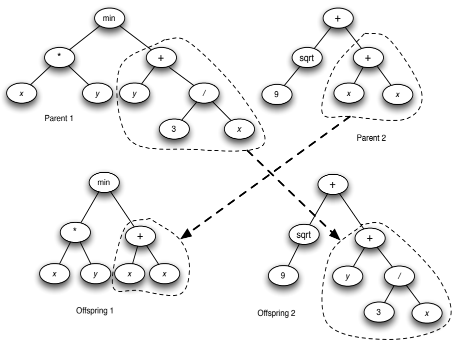


Figure 1.5: A crossover example of two parent trees producing two offsprings.

The first two steps define the search space that will be explored by GP. The fitness measure guides the search in promising areas of the search space and is a way of communicating a problem's requirements to a GP algorithm. The fitness evaluation of a particular individual is determined by the correctness of the output produced for all of the fitness cases [21]. The last two steps are administrative in their nature. The control parameters limit and control how the search is performed like setting the population size and probabilities of performing the genetic operations, while the termination criterion specifies the ending condition for the GP run and typically includes a maximum number of generations [46]. Genetic operators of mutation, crossover and reproduction are mainly responsible for introducing variation in successive generations. The crossover operator recombines randomly chosen parts from two selected programs and creates new program(s) for the new population (Figure 1.5). The mutation operator selects a point in a parent tree and generates a new random sub-tree to replace the selected sub-tree, while the reproduction operator simply replicates a selected individual to a new population.

The evolution of models using GP is an example of a symbolic regression problem. Symbolic regression represents one of the earliest applications of GP [157] and 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 [157]. In simpler terms, symbolic regression finds a function whose output matches some target values [208]. Throughout the thesis, whenever we refer to genetic programming for software engineering predictive modeling, we consider the symbolic regression application of it.

The next Subsection 1.2.3 presents an introduction to software fault prediction and discusses two classifications of quality evaluation models. Since many quality evaluation models exist in literature, these classifications help us relating them to the fault prediction studies in this thesis.

### 1.2.3 Software fault prediction

Errors, faults, failures and defects are inter-related terminologies and often have considerable disagreement in their definitions [81]. However, making a distinction between them is important and therefore for this purpose, we follow the IEEE Standard Glossary of Software Engineering Terminology [232]. 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 which, 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 [81] although it is a term more common in hardware and systems engineering [232]. 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 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 [82].

Software fault prediction models can be seen as belonging to a family of quality-evaluation models. As discussed in Section 1.1, these models may provide objective assessments and problem-area identification [238], 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 [106]. A fault prediction model uses previous software quality data in the form of software metrics to predict the number of faults in a module or release of a software system [144]. There are different types of models proposed in software verification and validation literature, all of them with the objective of accurately quantifying software quality. Different classifications of these models exists and we now discuss two of these classifications, one by Tian [238] and the other by Fenton and Neil [81].

#### Tian's classification of quality-evaluation models

This section serves as a summary of the classification approach given by Tian [238]. 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 of an overall model, a segmented model and a dynamic model:

- An overall model. Providing a single estimate of overall product quality, e.g. a single defect density estimate [125].
- 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 [212] which generalizes empirical effort and defect profiles over time into a Rayleigh curve[2].

Product-specific models are based on product-specific data. This type of models can further be divided into three types:

- 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 [171].
- Measurement-driven predictive models: Providing quality estimates using measurements from design and testing processes, e.g. [250].

### Fenton and Neil's classification of quality-evaluation models

Fenton and Neil [81] views the development of quality-evaluation models as belonging to four classes:

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

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

Prediction using size and complexity metrics represents majority of the fault prediction studies. Different size metrics have been used to predict the number of faults, e.g. Akiyama [12] and Lipow [165] used lines of code. There are also studies making use of McCabe's cyclomatic complexity [181], e.g. as in [153]. Then there are studies making use of metrics available earlier in the life cycle, e.g. Ohlsson and Alberg [197] used design metrics to identify fault-prone modules.

Prediction using testing metrics involves predicting residual faults by using faults found in earlier inspection and testing phases, e.g. [43]. Test coverage metrics have also been used to obtain promising results for fault prediction, e.g. [250].

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 [116].

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, e.g. [142, 186, 187].

### 1.2.4 Software fault prediction in this thesis

In relation to the classification schemes presented in Section 1.2.3, the software fault prediction studies in this thesis falls in the categories of observation-based models (with respect to Tian's classification) and predictions using testing metrics (with respect to Fenton and Neil's classification). This is depicted in Figure 1.6.

As discussed in Section 1.1, 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 [126, 127, 128, 129, 130] using neural network architectures for software reliability growth modeling. Other examples of studies reporting encouraging results include [5, 17, 70, 98, 99, 107, 134, 135, 136, 148, 228, 238, 239, 240, 241]. Apart from artificial neural networks, some authors have proposed using fuzzy models, as in [49, 50, 230, 249], and support vector machines, as in [242], to characterize software reliability. There are also studies that use a combination of techniques, e.g. [242], where genetic algorithms are used to determine an optimal neural network architecture and [193], where principal component analysis is used to enhance the performance of neural networks. The use of genetic programming for software fault prediction is reviewed in Chapter 4 of this thesis.

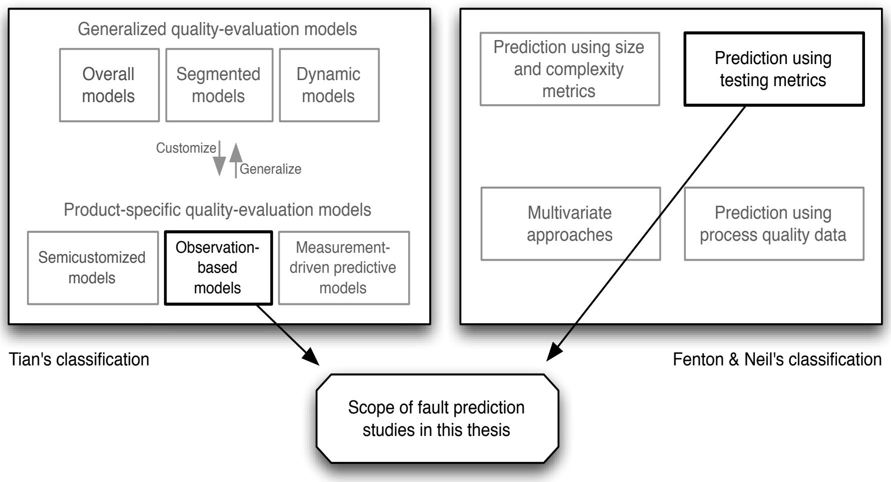

Generalized quality-evaluation models
Overall models
Segmented models
Dynamic models
Customize
Generalize
Product-specific quality-evaluation models
Semicustomized models
Observation-based models
Measurement-driven predictive models
Tian's classification
Prediction using size and complexity metrics
Prediction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

Figure 1.6: Relating fault prediction studies in this thesis to the two classification approaches.

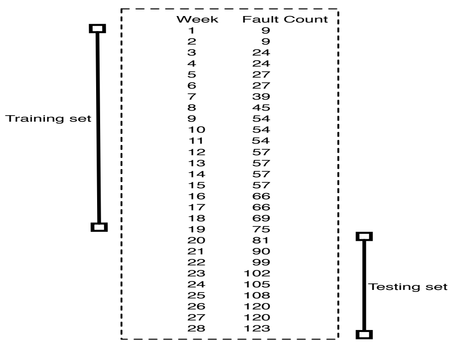
In relation to fault prediction studies in this thesis, it is useful to discuss some important constituent design elements. This concerns fault data sets, GP design and statistical hypothesis testing.

### Software fault data sets

The fault prediction studies in this thesis make use of fault data sets for two purposes:

1. To train the models using different techniques (the corresponding data set is called training set).
2. To test the trained models for evaluation purposes (the corresponding data set is called testing set).

The fault data sets used in this thesis resembles a time-series and represents weekly/ monthly faults gathered during the testing of various industrial and open-source projects. The data sets are impartially split into disjoint training and testing sets, with first 2/3 of the fault data set is used as training set while the later 1/3 of the data used as testing set. Further details about the fault data sets accompany the study details in Chapters 2


Training set
Testing set

| Week | Fault Count |
|---|---|
| 1 | 9 |
| 2 | 9 |
| 3 | 24 |
| 4 | 24 |
| 5 | 27 |
| 6 | 27 |
| 7 | 39 |
| 8 | 45 |
| 9 | 54 |
| 10 | 54 |
| 11 | 54 |
| 12 | 57 |
| 13 | 57 |
| 14 | 57 |
| 15 | 57 |
| 16 | 66 |
| 17 | 66 |
| 18 | 69 |
| 19 | 75 |
| 20 | 81 |
| 21 | 90 |
| 22 | 99 |
| 23 | 102 |
| 24 | 105 |
| 25 | 108 |
| 26 | 120 |
| 27 | 120 |
| 28 | 123 |

Figure 1.7: An example data set split into training and testing sets.

and 3 of this thesis. Figure 1.7 represents one example data set split into training and testing sets.

### Genetic programming design

For studies in this thesis, we have one independent variable $x$ (week/month number) making up the terminal set. It is also common to complement the terminal set with randomly generated constants within a suitable range; however the choice of these constants is problem-dependent. For the studies in this thesis, the terminal set is taken to only contain the independent variable, i.e. $T$=$\{x\}$.

The choice of function sets is also problem-dependent; however, ordinary arithmetic functions are normally used for numeric regression problems [208]. The studies in this thesis use different function sets for different data sets, one example being, $F = \{+, -, *, sin, cos, log\}$.

The quality of solutions is measured using an evaluation measure. We use a natural evaluation measure for symbolic regression problems which is the calculation of the difference 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 last two steps of the GP design are administrative and concerns setting the parameters for a GP run. This includes selecting population size, setting number of generations, tree-initialization method, selection method and any restrictions on the size of program trees. The selection of these parameters is, yet again, problem de-

Table 1.1: Example control parameters used for the GP system.

| *Control Parameter* | *Value* |
|---|---|
| Population size | 30 |
| Number of generations | 200 |
| Termination condition | 200 generations |
| Function set | $\{+, -, *, sin, cos, log\}$ |
| Terminal set | $\{x\}$ |
| Tree initialization | ramped half-and-half [3] |
| Initial maximum number of nodes | 28 |
| Maximum number of nodes after genetic operations | 512 |
| Genetic operators | crossover [4], mutation [5], reproduction [6] |
| Selection method | lexictour [7] |
| Elitism | replace [8] |

[1]Balanced and unbalanced trees of different depths.
[2]Branch swapping by randomly selecting nodes of the two parent trees.
[3]A random node from the parent tree is substituted with a new random tree.
[4]Copy of trees to the next generation without any operation.
[5]Selecting a random number of individuals from the population and choosing the best of them,
if two individuals are equally fit, the one having the less number of nodes was chosen as the best.
[6]Children replace the parent population having received higher priority of survival,
even if they are worse than their parents.

pendent and Chapters 2 and 3 in this thesis present the parameter settings for the GP algorithm in this case. However, for the sake of completeness Table 1.1 shows one example of control parameter settings for a GP algorithm.

### Evaluation measures and statistical hypothesis testing

Statistical hypothesis testing is used to test a formally stated null hypothesis and is a key component of the analysis and interpretation phase of experimentation in software engineering [258]. 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[3]. Chapters 2 and 3 in this thesis make use of statistical hypothesis testing to draw conclusions.

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

[3]Simply relying on statistical calculations is not always reliable either, as was clearly demonstrated by Anscombe in [20] where he showed the necessity of actually looking at plotted data.

could have been caused by chance alone [190]. The use of statistical tests of significance comes with its own share of challenges about which tests are suitable for a given problem. A study by Demšar [66] recommends non-parametric (distribution free) tests for statistical comparisons of classifiers; while elsewhere in [34] 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, 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 [256].

Prior to applying statistical testing, suitable accuracy indicators are required. However, there is no consensus with regards as to which accuracy indicator is the most suitable for the problem at hand. Commonly used indicators suffer from different limitations [85, 223]. 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 [85]. In [195], measures for the following characteristics are proposed: 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 [87, 177], average relative error is used as a measure of variable-term predictability.

As an example of applying multiple measures, the study in Chapter 2 of this thesis use 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 distribution of residuals is proposed as an alternative measure [149, 223]. It has the convenience of applying significance tests and visualizing differences in absolute residuals of competing models using box plots.

We 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 empirical evaluation as an insufficient way to judge a model's output accuracy. Qualitative criterion-based evaluation judges each method based on conceptual requirements [90]. One or more of these requirements might influence model selection. The study in Chapter 3 presents such qualitative criteria based evaluation, in addition to quantitative evaluation.

The next Subsection 1.2.5 describes the concept of systematic literature reviews and how they are applicable in this thesis.

### 1.2.5 Systematic literature reviews

A systematic review evaluates and interprets all available research relevant to a particular research question [150]. The aim of the systematic review is therefore to consolidate all the evidence available in the form of primary studies. A systematic review is at the heart of a paradigm called evidence-based software engineering [77, 119, 151] which is concerned with objective evaluation and synthesis of best quality primary studies relevant to a research question. A systematic review differs from a traditional review in following ways [244]:

- 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 [150] 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 1.8 and summarized in the following paragraph:

1. *Identification of the need for a review*—the reasons for conducting the review.

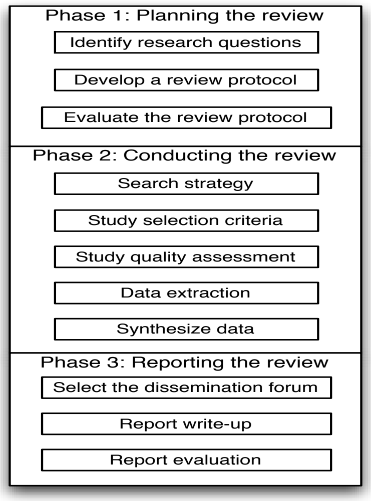

Phase 1: Planning the review
Identify research questions
Develop a review protocol
Evaluate the review protocol
Phase 2: Conducting the review
Search strategy
Study selection criteria
Study quality assessment
Data extraction
Synthesize data
Phase 3: Reporting the review
Select the dissemination forum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

Figure 1.8: The systematic review stages.

2. *Research questions*—the topic of interest to be investigated e.g. assessing the effect of a software engineering technology.
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.

### 1.2.6 Systematic reviews in this thesis

This thesis contains two systematic reviews making up Chapters 4 and 5. The systematic review in Chapter 4 consolidates the application of symbolic regression using

GP for predictive studies in software engineering. There were two major reasons for carrying out this study:

1. To be able to draw (if possible) general conclusions about the extent and effectiveness of application of symbolic regression using GP for predictions and estimations in software engineering.
2. To summarize the benefits and limitations of applying symbolic regression using GP as a prediction and estimation tool.

The systematic review in Chapter 5 examines the existing work in search-based testing of non-functional system properties. The focus of this systematic review is on non-functional testing, since it was evident from an earlier survey paper on search-based software test data generation by McMinn [182] that search-based non-functional software testing is a field having potential but lacking empirical results. McMinn [182] also included suggestions in his paper about the possibility of non-functional properties being tested using search-based techniques. Ever since the publication of McMinn's survey paper, it has been important and interesting to know the extent of which search-techniques has been applied to non-functional testing. The systematic review in Chapter 5 investigated the literature into non-functional properties being tested using search-based techniques (answering research question 4, Section 1.3).

Finally, the motivations for carrying out this study were as follows:

1. To be able to identify existing non-functional properties being tested using search techniques.
2. To identify any constraints and limitations in the application of these search techniques.
3. To identify the range of fitness functions used in the application of these search techniques and, in cases where possible, to present an analysis of these fitness functions.

## 1.3 Research questions and contribution

The specific purpose and goals of research in general are very often highlighted in the form of specific research questions [63]. These research questions relate to one or more main research question(s) that clarifies the central direction behind the entire investigation [63].

The purpose of this thesis is to determine the applicability of search-based techniques in two activities within software verification and validation: Software predictive modeling and software testing. The main research question of the thesis is thus based on this purpose and is formulated as:

*Main Research Question:* What is the applicability of search-based techniques for software verification and validation in the context of software predictive modeling and software testing?

This main research question highlights the two investigative components of the thesis within software verification and validation, i.e., software predictive modeling and software testing, the common denominator being the application of search-based techniques. The main research question is further divided into two sub-questions addressing the two components. These two sub-questions are formulated as below:

*Research sub-question 1:* What is the trade-off in using search-based techniques for software engineering predictive modeling with a focus on software fault prediction?

*Research sub-question 2:* What is the current state of research considering testing of non-functional system properties using search-based techniques?

The above two research sub-questions are answered by posing specific research questions which are addressed in one or more chapters forthcoming in the thesis.

There are two specific research questions answering *research sub-question 1*, RQ1.1. and RQ1.2. The first specific research question RQ1.1. is formulated below:

*RQ1.1.* What is the quantitative and qualitative performance of genetic programming in modeling fault count data?

*Related concepts:* Search-based software engineering (SBSE) (Section 1.2.1), genetic programming (GP) (Section 1.2.2), software fault prediction (Section 1.2.3).
*Relevant chapters:* Chapters 2 and 3.

Chapter 2 serves as a stepping-stone for the research into search-based software fault prediction. Chapter 2 constitutes a sequential multi-step value-addition. Specifically, the first step discusses the mechanism enabling genetic programming to progressively search for better solutions and potentially be an effective prediction tool. The second step explores the use of genetic programming for software fault count predictions by evaluating against five different 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. Thus the overall contributions of the chapter are: (i) Exploring the GP mechanism that might be suitable for modeling (ii) Empirically investigating the use of GP as a potential prediction tool in software V&V (iii) Comparative evaluation of using GP with traditional software reliability growth models (iv) Evaluating earlier published results using GP as a prediction tool.

The early positive results of using GP for fault predictions in Chapter 2 warranted further investigation into this area which, resulted in the write-up of Chapter 3. Chapter 3 investigates *cross-release* prediction of fault data from large and complex industrial and open source software. The comparison groups, in addition to using symbolic regression in genetic programming, include both traditional and machine learning models, while the evaluation is done both quantitatively and qualitatively. The overall contribution of the chapter is therefore quantitative and qualitative assessment of the generalizability and real-world applicability of different models for cross-release fault predictions using extensive data sets covering both open source and industrial software projects.

The second specific research question (RQ1.2.) answering *research sub-question 1* takes a step back from software fault prediction and presents a broader perspective on the application of search-based techniques. This broader perspective is in terms of addressing not only software fault prediction but also prediction of other attributes within software engineering:

*RQ1.2.* 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?

*Related concepts:* Search-based software engineering (SBSE) (Section 1.2.1), genetic programming (GP) (Section 1.2.2), software fault prediction (Section 1.2.3), systematic literature reviews (Section 1.2.5)
*Relevant chapter:* Chapter 4.

RQ1.2. is answered using a systematic literature review investigating the extent of application of symbolic regression in genetic programming within software engineering predictive modeling (Chapter 4). The purpose of carrying out this review is discussed in Section 1.2.6. Besides being a systematic review answering the posed research question, other contributions of the chapter include:

- Presenting an opportunity to assess different attributes that can be measured using GP and its efficacy.

- An understanding of different GP variations used by the review studies to predict and estimate in a better way.

Figure 1.9 shows the relation between different specific research questions connected to the *research sub-question 1*.

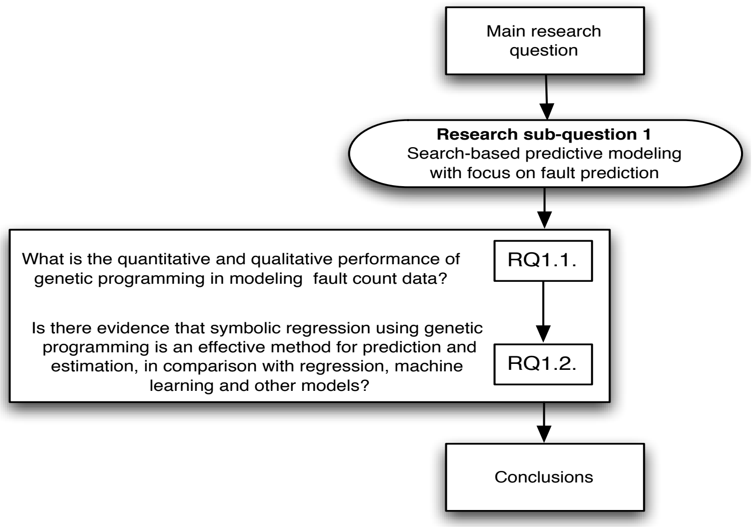

Main research question
Research sub-question 1
Search-based predictive modeling with focus on fault prediction
What is the quantitative and qualitative performance of genetic programming in modeling fault count data?
RQ1.1.
Is there evidence that symbolic regression using genetic programming is an effective method for prediction and estimation, in comparison with regression, machine learning and other models?
RQ1.2.
Conclusions

Figure 1.9: Relationship of specific research questions with the research sub-question 1 and the main research question.

The *research sub-question 2* is answered by posing the following specific research question addressed in Chapter 5 of this thesis.

*RQ2.1.* In which non-functional testing areas have metaheuristic search techniques been applied?

*Related concepts:* Search-based software engineering (SBSE) (Section 1.2.1), systematic literature reviews (Section 1.2.5)
*Relevant chapter:* Chapter 5.

RQ2.1. is further divided into the following research sub-questions:

*RQ2.1.1.* What are the different metaheuristic search techniques used for testing each non-functional property?
*RQ2.1.2.* What are the different fitness functions used for testing each non-functional property?
*RQ2.1.3.* What are the current challenges or limitations in the application of metaheuristic search techniques for testing each non-functional property?

RQ2.1. (Chapter 5) is answered through a systematic literature review of application of search-based techniques for non-functional testing. Besides the purpose of this systematic review being discussed in Section 1.2.6, the contribution of the chapter is an exploration of non-functional properties tested using search-techniques, identification of constraints and limitations encountered and an analysis of different fitness functions used to test individual non-functional properties.

The relationship between the specific research question RQ2.1. and the associated sub-questions is depicted in Figure 1.10.

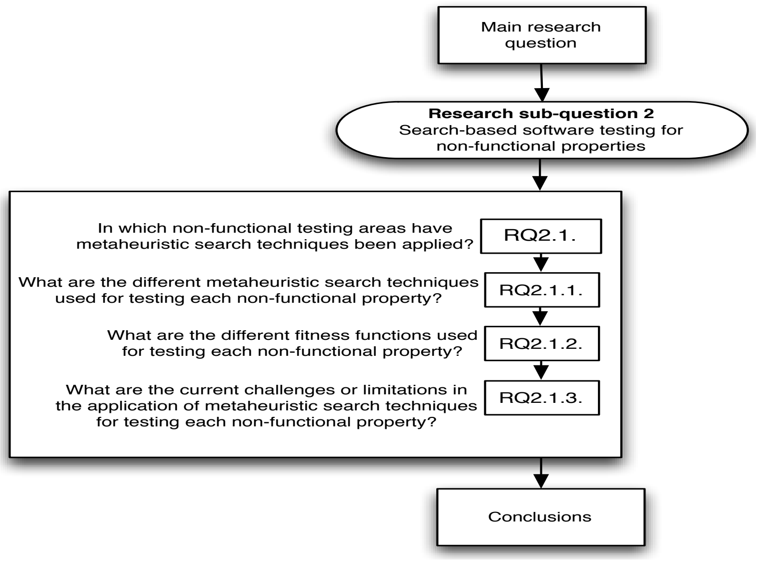

Main research question
Research sub-question 2
Search-based software testing for non-functional properties
In which non-functional testing areas have metaheuristic search techniques been applied?
RQ2.1.
What are the different metaheuristic search techniques used for testing each non-functional property?
RQ2.1.1.
What are the different fitness functions used for testing each non-functional property?
RQ2.1.2.
What are the current challenges or limitations in the application of metaheuristic search techniques for testing each non-functional property?
RQ2.1.3.
Conclusions

Figure 1.10: Relationship of specific research questions with the research sub-question 2 and the main research question.

The two research sub-questions therefore have a focus on search-based software predictive modeling and search-based software testing. Figure 1.11 presents a high-level view on the relationship between the main research question, the research sub-questions, the specific research questions and the related concepts. Figure 1.11 shows that the main research question has two major concerns i.e. application of *search-based techniques* within *software verification and validation*. We have a focus on two activities within software verification and validation: Software predictive modeling and software testing. Our research sub-questions are formulated based on these two activities. The research sub-questions are answered by three specific research questions: RQ1.1., RQ1.2 and RQ2.1. The specific research questions are subjects of subsequent Chapters 2, 3, 4 and 5 of the thesis, allowing us to draw conclusions regarding the main research question.

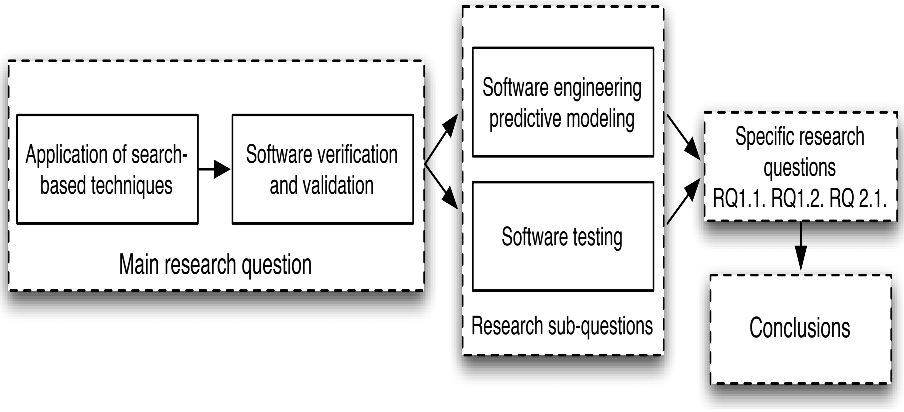

Application of search-based techniques
Software verification and validation
Main research question
Software engineering predictive modeling
Software testing
Research sub-questions
Specific research questions RQ1.1. RQ1.2. RQ 2.1.
Conclusions

Figure 1.11: A high-level view on the relationship between the different research questions and the concepts.

## 1.4 Research methodology

Research approaches can usually be classified into quantitative, qualitative and mixed methods [63]. 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 [63]. A qualitative approach to research, on the other hand, is based on theory building relying on human perspectives. This approach accepts that there are different ways of interpretation [258]. The mixed methods approach involves using both quantitative and qualitative approaches in a single study.

The below text takes a tour of different strategies associated with quantitative, qualitative and mixed method approaches [63]. In the end, the relevant research methods for this thesis are discussed.

### 1.4.1 Qualitative research strategies

Ethnography, grounded theory, case study, phenomenological research and narrative research are examples of some qualitative research strategies [63].

*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 [63]. *Grounded theory* is 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 [63]. A *case study* involves in-depth investigation of a single case, e.g. an event or a process. The case study has time and work limits within which different data collection procedures are applied [63]. *Phenomenological research* is grounded in understanding the human experiences concerning the phenomenon [63]. Like in ethnography, phenomenological research involves prolonged engagement with the subjects. *Narrative research* is akin to retelling stories about other individuals' lives and relating with researcher's life in some manner [63].

### 1.4.2 Quantitative research strategies

Quantitative research strategies can be divided into two quantitative strategies of inquiry [63]: Experiments and surveys.

An experiment, or "[. . . ] a formal, rigorous and controlled investigation" [258], 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 [63]. *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 [63].

Robson, in his book *Real World research* [218], 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 include 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 collects 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.

### 1.4.3 Mixed method research strategies

The mixed method research strategies can use sequential, concurrent or transformative procedures [63]. 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 [63]. *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 [63].

With respect to specific research strategies, surveys and case studies can be both quantitative and qualitative [258]. 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 1.2.5) as a form of survey. A systematic literature review can also be quantitative or qualitative depending on the data synthesis [150]. Using statistical techniques for quantitative synthesis in a systematic review is called meta-analysis [150]. However, software engineering systematic literature reviews tend to be qualitative (i.e. descriptive) in nature [32]. 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 [152].

### 1.4.4 Research methodology in this thesis

The chapters in this thesis are based on both quantitative and qualitative research methodologies. Chapters 2 and 3 of this thesis fall within the category of prediction studies (Section 1.4.2) and thus belonging to the high-level category of non-experimental fixed designs. Specifically, the studies constituting Chapters 2 and 3 make use of a predictor variable (week/month number) to predict the criterion variable (fault counts). Also these studies use quantitative data collected over time which is used both for training the models and testing the predictions (Section 1.2.4). Chapter 3 is additionally complemented with a qualitative assessment of models so it is justifiable to place it under a mixed methods approach using sequential procedure. Chapters 4 and 5 are systematic reviews and since they include descriptive data synthesis, these are the candidates for qualitative studies. Table 1.2 presents the research methodologies used in this thesis in tabular form.

Table 1.2: Research methodologies used in this thesis.

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

## 1.5 Validity evaluation

Experimental results can be said to have sufficient validity if they are valid for the population under interest [258]. This validity is compromised due to threats against four types of validity i.e. conclusion, internal, construct and external validity [258].

*Conclusion validity* is related to the strength of the relationship between the treatment and the outcome [258]. In our studies in Chapters 2 and 3, we have used statistical hypothesis testing at commonly used significance levels of 0.01 and 0.05 to identify any significant relationships between the observed and the predicted data. Generally, we were conscious of the assumptions of the statistical tests and used the type of tests (parametric or non-parametric) accordingly. The selection of evaluation measures is also another threat to conclusion validity because there is still a lack of clear consensus on the most suitable evaluation measures to use. This threat is minimized in two ways; first objective measures are applied [258] and secondly more than one measure is applied to cross-check the results if possible.

The conclusion validity threats in case of the systematic reviews in Chapters 4 and 5 are different from standard validity threats being flexible design types rather than the fixed ones in case of experiments. One conclusion validity threat in case of these studies is bias in applying quality assessment and data extraction. This is minimized by basically following the guidelines for conducting systematic reviews [150] which makes it explicit how quality assessment and data extraction are carried out. Specifically for checking the consistency of data extraction, a small sample of primary studies was used to extract data for the second time.

*Internal validity* is related to causality i.e. the relationship between the treatment and the outcome should be a causal one [258]. The studies in Chapters 2 and 3 use different ways to minimize the threats against internal validity. 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 [257]. 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. This aggregation of data may have consequences since it does not capture the *effective* work hours during each week/month. This situation could have been improved by collecting data from an ongoing project in an online context, rather than using a complete historical set upfront. This is intended to be part of our future studies.

For the systematic review studies in Chapters 4 and 5, one threat to internal validity arises from not including research that remained unpublished due to undesirable results or proprietary literature that is not made available. Although it is difficult to find such grey literature, some more effort in this regard would have improved the internal validity of the results.

*Construct validity* is concerned with generalizing the experimental results to the theory behind the experiment [258]. In our studies in Chapters 2 and 3, those evaluation measures were used that relate to the measurement of a specific property, or to put it in other words, reflects the construct under study [258] e.g. the Kolmogorov-Smirnov test is used for measurement of goodness of fit test which is a commonly used test for this measure. Moreover, the number of data sets reflected a reasonable representation of treatments in our opinion with three industrial data sets used in Chapter 2 and a total of seven data sets (industrial and open-source) used in Chapter 3.

For the systematic reviews in Chapters 4 and 5, a threat to construct validity could be that we missed relevant studies. This threat however was minimized by using a thorough, well-defined and constantly refined search strategy.

*External validity* is concerned with the generalization of the results outside the scope of the study [258]. While Chapter 2 involves data sets from projects undertaken by one organization, Chapter 3 includes data sets from diverse projects from different software organizations and open-source projects, hence helping to improve the generalizability of results achieved.

For the systematic reviews in Chapters 4 and 5, the external validity can be related to the degree to which the primary studies are representative of the overall goal of the review. This is covered by the systematic review protocol, which helped us to achieve a representative set of studies to a greater extent.

## 1.6 Summary

In this chapter, we presented a synopsis of the research area and what we believe to be the contribution of the research. We additionally presented the concepts that will be used in later chapters of this thesis and outlined the research methodology used along with the validity evaluation. The next Chapter 2 presents findings and conclusions of using genetic programming as a potential prediction tool in software V&V.

# Chapter 2

# Genetic programming for software fault count predictions

## 2.1 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 [232]. If the software frequently fails to perform according to user-specified behavior, other software quality factors matters less [188]. 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. An 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 [192]. The model then estimates reliability or predicts future reliability by different forms of extrapolation [172]. After the first software reliability growth model was proposed by Jelinski and Moranda in 1972 [115], there have been numerous reliability growth models following it. These models come under different classes [171], 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 it 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 [195]. 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 in real-world situations (see e.g. [260]), 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 as 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 2.2 and Section 2.3 present related work and a background to genetic programming, respectively. Section 2.4 discusses stage one of the study. The second study stage is discussed in Section 2.5 and consists of a discussion on the research method, experimental setup, results and summary of results. The third study stage is discussed in Section 2.6, 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 2.7 while the chapter ends with a discussion and conclusions in Section 2.8 and Section 2.9, respectively.

## 2.2 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. [228]), 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. [61] 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 [200] and reported improved results.

A similar study by Zhang and Chen [266] 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.

## 2.3 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 [157]. 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 [229] 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. GP 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 [208]. 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 [21]. 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 [46]. 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.

## 2.4 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 [157], the GP population contains building blocks, which could be any GP tree or sub-tree in the population. 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 [23]. This hypothesis appears suited to adaptive model-building system that can be used for predicting software reliability growth.

The evolution of better individuals using GP is shown in Figure 2.1.

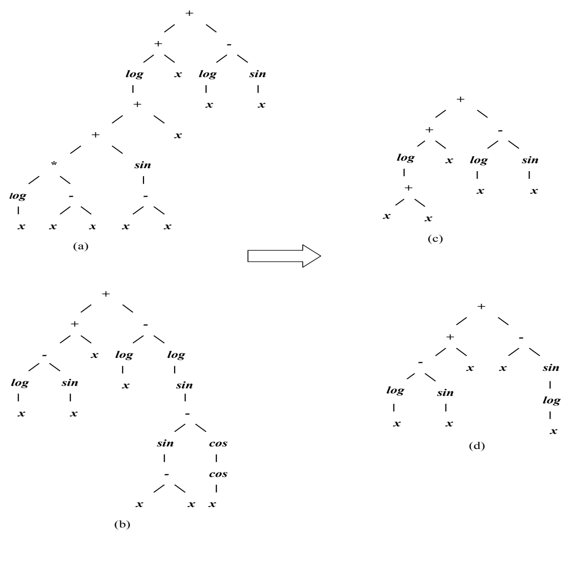


Figure 2.1: Combination of trees containing building blocks.

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 2.1a and 2.1b in Figure 2.1) containing different building blocks for an optimum solution. For tree 1 (Figure 2.1a), the sum of absolute differences between the obtained and expected results in all fitness cases was 31.34, while for tree 2 (Figure 2.1b), the fitness measure was 28.9. By combining these two trees, two new trees could emerge (Figure 2.1c and Figure 2.1d). The first tree (Figure 2.1c) has a better fitness of 27.8 than any of its parents, while the second tree (Figure 2.1d) 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 to be able to comply with all the fitness cases [208]. Evolutionary algorithms have been found to be robust for complex search spaces. Genetic programming can potentially be a valid technique to evolve software reliability growth model 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 [208]. Figure 2.2 shows how the GP algorithm is searching the program space of solutions to track the model to approximate.

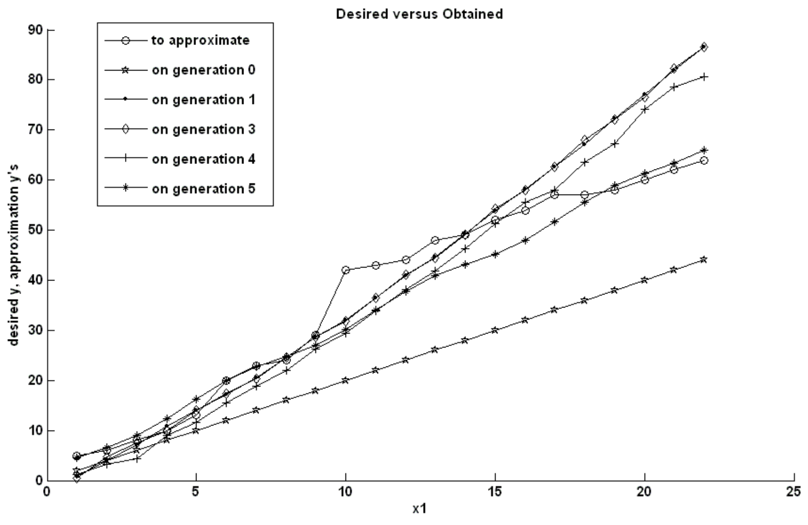


Figure 2.2: Several approximations to the original fault count data in different generations.

Figure 2.3 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 [184]. The Pareto front in Figure 2.3 shows the set of solutions for which no other solution was found which both had a smaller tree and better fitness [227]. The Figure 2.3 also shows the best fitness found for each tree size. It is clear from Figure 2.3 that the fitness of different solutions fluctuates as the number of nodes increases during the course of generations.

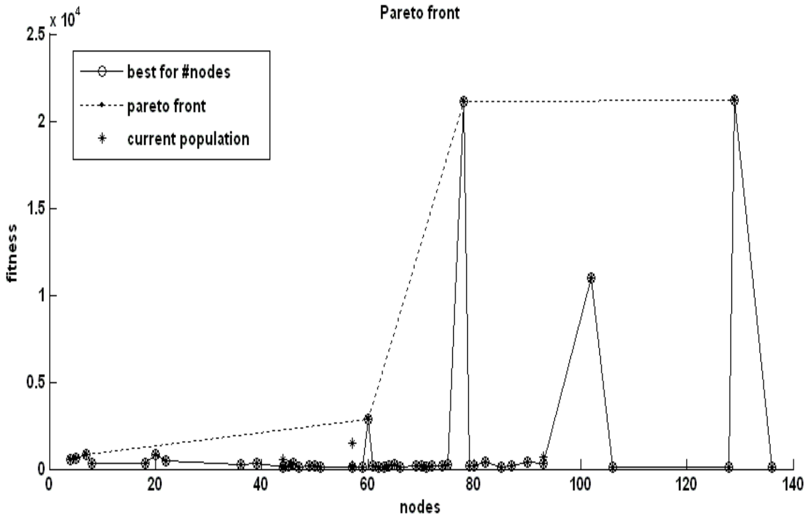


Figure 2.3: Visualization of Pareto front for one set of industrial fault count data.

## 2.5 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.

### 2.5.1 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 followed an iterative development process meaning that within each iteration, a new system version, containing new functionality and fixes of previously discovered faults, was delivered to test. These iterations occurred on a weekly basis or even more frequently, while testing of new releases proceeded 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 2.5.1). 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:

$H_{0-\text{acc}}$: GP model does not produce significantly accurate predictions. $H_{1-\text{acc}}$: GP model produces significantly accurate predictions. $H_{0-\text{gof}}$: GP model does not fit significantly to a set of observations. $H_{1-\text{gof}}$: 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 [195], because reliance on a single measure can lead to making incorrect choices. The deviation between observed and the fitted values was, 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 will they be used later in the study.

**Kolmogorov-Smirnov** The K-S test is a commonly used statistical test for measuring goodness of fit [234, 180]. 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.

$$H_0 : [F(t) = G(t), \text{for every } t] \tag{2.1}$$

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 $J_\alpha$, 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 [109].

**Spearman's rank correlation coefficient** Spearman's rank correlation coefficient $\rho$ 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 $\rho$ exceeds the critical values of $\rho$ 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 rank correlation coefficient can be found in [122].

**Prediction at level $l$** Prediction at level $l$, $pred(l)$, 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 [71].

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

Conte et al. [58] consider MMRE $\leq 0.25$ 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 [259] for prediction stability which says that a prediction is stable if the prediction in week $i$ is within 10% of the prediction in week $i-1$.

### 2.5.2 Experimental setup

In this study we used MATLAB version 7.0 [179] and GPLAB version 3.0 [227] (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,

$$\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid \tag{2.2}$$

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 2.1.

### 2.5.3 Results

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

Table 2.1: 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 |

## Adaptability of the model

Table 2.2 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 $J_\alpha$ for $\alpha = 0.05$ are also given. We selected the significance level ($\alpha$) of 0.05 as it is common in practice [121]. 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 2.2: 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 $\rho$ for determining the relationship between actual and estimated model values (Table 2.3). At significance level $\alpha = 0.05$, computed values of $\rho$ 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.

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

Table 2.3: Results of applying Spearman's correlation coefficient test.

| | $\rho$ | $r_{\alpha=0.05}$ | *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 | $\surd$ |

alternative hypothesis, $\mathrm{H}_{1-\mathrm{gof}}$.

**Measuring predictive accuracy**

Table 2.4 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. In all the data sets, the measurement $pred(0.25) \geq 0.75$ holds true. The bold values in Table 2.4 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 $\leq 0.25$, 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 2.5) indicate that the predictions are indeed stable.

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, $\mathrm{H}_{0-\mathrm{acc}}$ in favor of the alternative, $\mathrm{H}_{1-\mathrm{acc}}$.

Figure 2.4 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.

## 2.5.4 Summary of results

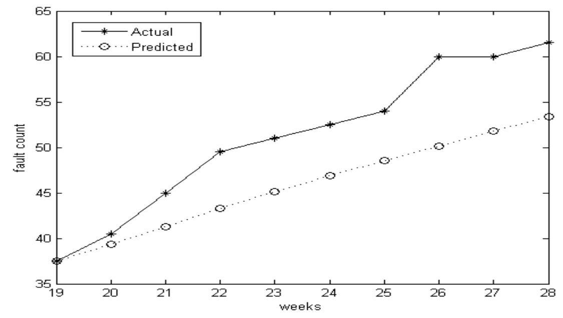
The hypothesis to be tested was that GP could be a suitable approach for evolving an SRGM based on fault count data. The results of applying the evaluation criteria, as described in Section 2.5.1, confirmed that GP represents a suitable approach for modeling software reliability growth based on fault count data, both in terms of goodness of fit

Table 2.4: Testing for $\mathbf{pred}(\mathbf{0.25}) \geq \mathbf{0.75}$.

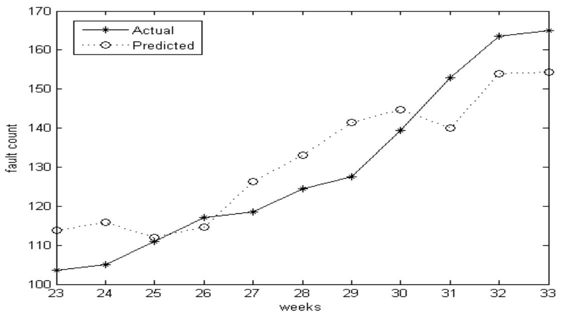
| *Week i* | $25\%\ of\ e_i$ | $e_i^{'}$ | $e_i^{'}$ *within range of* $25\%\ of\ e_i$? |
|---|---|---|---|
| Project 1 | | | |
| 19 | $25 \pm 6.25$ | 25 | $\surd$ |
| 20 | $27 \pm 6.75$ | 26.23 | $\surd$ |
| 21 | $30 \pm 7.5$ | 27.53 | $\surd$ |
| 22 | $33 \pm 8.25$ | 28.83 | $\surd$ |
| 23 | $34 \pm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

Table 2.5: Testing for prediction stability.

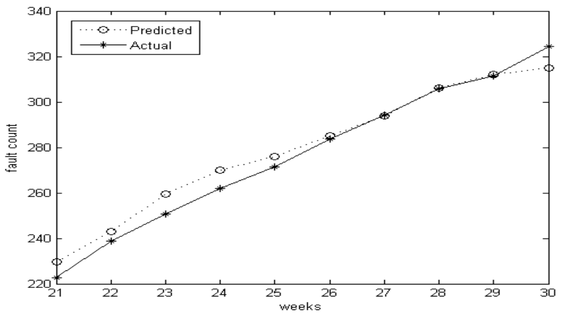
| *Week i* | *Prediction in week i* | 10% *of the prediction in week* $i-1$ | *Prediction stability* |
|---|---|---|---|
| | | Project 1 | |
| 19 | 25 | – | – |
| 20 | 26.23 | $25 \pm 2.5$ | $\surd$ |
| 21 | 27.53 | $26.23 \pm 2.62$ | $\surd$ |
| 22 | 28.83 | $27.53 \pm 2.75$ | $\surd$ |
| 23 | 30.10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



(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 2.4: Actual and predicted fault count data for three projects.

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 $\leq 0.25$ 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 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 (Section 2.5.1) which also gave results in support of GP.

## 2.6 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 selection of traditional SRGMs, hypotheses, the evaluation measures and the results. We do not discuss the experimental set up as it was the same as for stage 2 (Section 2.5.2).

### 2.6.1 Selection of traditional SRGMs

Since we are interested in comparing predictions of weekly fault count data, therefore we selected three traditional SRGMs that represent the fault count family of models [88]. These three models are Goel-Okumoto non-homogeneous Poisson process model (GO-NHPP) [89], Brooks and Motley's Poisson model (BM) [40] and Yamada's S-shaped growth model (YAM) [261]. We selected them because these models present a fair representation of fault count family of models and 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 exponential model's hazard function requires knowing the parameters of the total number of instructions in the program and debugging time since the start of system integration [88].

### 2.6.2 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.

### 2.6.3 Evaluation measures

It is usually recommended to use more than one measure to determine model applicability (see e.g. [195]), because reliance on a single measure can lead to making incorrect choices. We used measures of model validity, model 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 [39]:

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

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 trustworthy predictions. For further details on PLR, see [3, 39]. 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 give by the following formula [39]:

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

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 [45]:

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

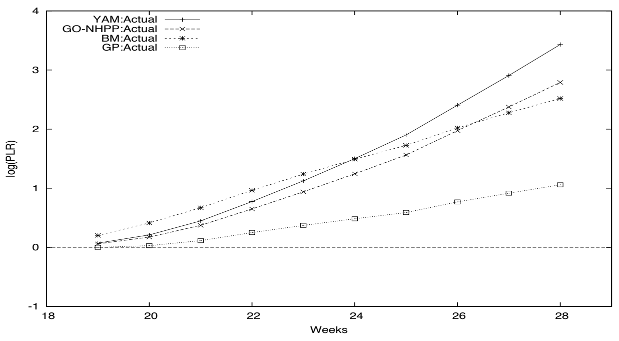
where $E_i$ is the actual fault count data and $\hat{E}_i$ is the predicted fault count data.

To measure a particular model's bias, we examine the *distribution of residuals* to compare models as suggested in [149, 205]. The model's *goodness of fit* in our case was measured using the Kolmogorov-Smirnov (K-S) test [109]. 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 $J_\alpha$, we infer that the two samples did not have the same probability distribution and hence do not represent significant goodness of fit.

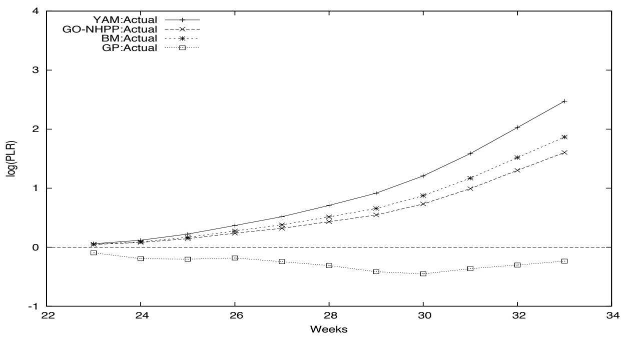
### 2.6.4 Results

Figure 2.5 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 2.5. 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 modeling reality as compared with traditional reliability growth models.

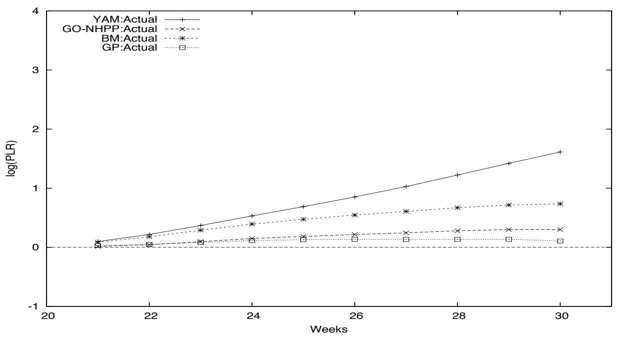
The variability measures of Braun statistic and AMSE obtained for each data set of all models were compared using matched paired two-sided $t$-test at significance level, $\alpha$ = 0.1. We compared the variability measures of the GP model with each of the traditional models. The null hypothesis was 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



(a) Log(PLR) plots for Project 1.



(b) Log(PLR) plots for Project 2.



(c) Log(PLR) plots for Project 3.

Figure 2.5: Log(PLR) plots for three projects.

Table 2.6: Statistical results for Braun statistic and AMSE.

| *Comparative models* | *t-statistic* |
|---|---|
| Braun statistic, $t_\alpha$=±2.42 | |
| GP:BM | −3.97 |
| GP:YAM | −4.80 |
| GP:GO-NHPP | −1.64 |
| AMSE statistic, $t_\alpha$=±2.42 | |
| GP:BM | −1.23 |
| GP:YAM | −1.39 |
| GP:GO-NHPP | −1.03 |

approximately normal distribution. The results of applying the matched paired two sided $t$-test are shown in Table 2.6.

The critical values of $t$ for $\alpha$=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 2.6 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 optimizes the Braun statistic, degrades AMSE. This result strengthens the viewpoint of Mair et al. [175] 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, $H_{0-\text{val}}$ in support of the alternative hypothesis, $H_{1-\text{val}}$.

Table 2.7 shows the statistic $J$ for the two sample K-S test performed on the validation fault count data ($1/3$ of the original data set) and the predictions by the GP and traditional reliability growth models. 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 $J_\alpha$. For project 2, the GP model along with GO-NHPP model have K-S statistic $J$ less than $J_\alpha$; and for project 3, the GP model along with BM and GO-NHPP provide K-S statistic $J$ less than $J_\alpha$.

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

Table 2.7: Results of applying K-S test.

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

Table 2.8: **t**-test results for residuals.

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

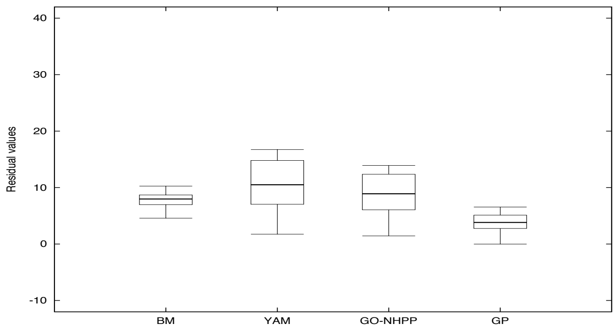
reject the null hypothesis, $H_{0-\text{gof}}$ so we are inconclusive regarding the significance of the goodness of fit of competing models.

Figure 2.6 shows the box plots of the residuals for all the models for the three projects. For project 1 (Figure 2.6a), all the box plots show the tendency of under-estimating; 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 2.6b) is to overestimate but the bias is smaller as compared to other models. In case of project 3 (Figure 2.6c), all box plots represent a tendency to under-estimate while the GP model presents relatively less bias with residuals both above and below 0.

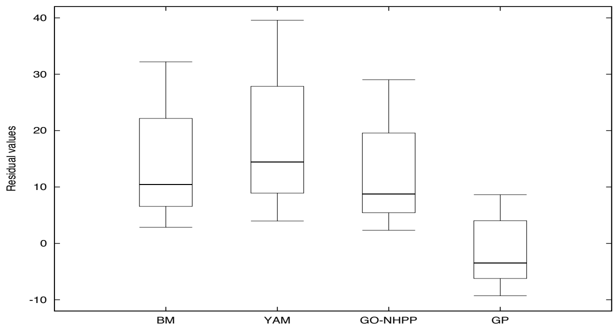
Since the box plots in Figure 2.6 are not significantly skewed, we applied matched paired $t$-tests of the residuals for each data set to compare the GP model with each of the traditional models. The results are presented in Table 2.8 and show that the residuals from the GP model are significantly different and less variable from the residuals for traditional models for each data set at $\alpha$=0.05. Therefore, we are able to reject the null hypothesis, $H_{0-\text{res}}$ in support of the alternative hypothesis, $H_{1-\text{res}}$.

### 2.6.5 Summary of results

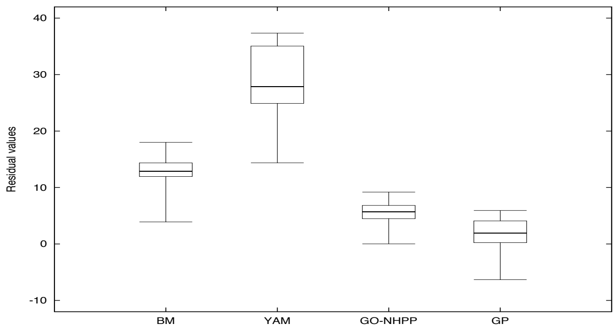
Stage 3 of the study presented the results of comparative evaluation of fault count data predictions from models evolved by genetic programming and traditional reliability growth models. The results have been evaluated in terms of model validity, goodness of fit and distribution of residuals. For evaluating model validity, the results of using prequential likelihood ratio show favorability concerning the GP model. However, the



(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 2.6: Charts showing box plots of residuals for three projects.

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.

## 2.7 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 between the treatment and the outcome [258]. One of the threats to conclusion validity is the use of MMRE in stage 2 of the study which has been criticized in [85] 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 the authors' 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) [258]. 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 [257]. 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 [258]. 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.

## 2.8 Discussion

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

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 [208].

While measures of goodness of fit and predictive accuracy are important, we agree with Mair et al. [175] that these measures are not enough for a practical utility of a 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 GP 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 [208].

## 2.9 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 three 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 of 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 3, which investigates cross-release prediction of fault-count data from large and complex industrial and open source software.

# Chapter 3

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

## 3.1 Introduction

Software is playing an increasingly important role in society and software development organizations are striving for cost-effective development and time-to-market to remain competitive. At the same time, the time to develop software and the size and complexity of software is increasing, so there are many challenges in delivering quality software on time and within stipulated cost. 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% [27] and, hence, efficient resource allocation for quality assurance activities is required. Thus, fault prediction models have attracted considerable interest (as shown in Section 3.2), 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 project 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 module, or in a particular release of a 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 module or a release [144]. 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 size [90]. Therefore, it is not surprising that we possess an incomplete understanding of the phenomenon under study since *it is very difficult to make valid assumptions about the form of the functional relationship between the variables* [33]. 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 [90]. Applications of computational and artificial intelligence have attempted to deal with some of these challenges, see e.g. [265], 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.

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

2. They are independent of any assumptions about the stochastic behavior of software failure process and the nature of software faults [235].
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. [240]. 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 (covered in Section 3.2). Our main focus is on evaluating a variety of techniques for cross-release prediction of fault counts, on data sets from large real world projects; to our knowledge this is novel. We evaluate the created models on fault data from several large and real world software projects, some from open-source and some from industry (see Section 3.3).

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 [88]. These three models are Goel-Okumoto non-homogeneous Poisson process model (GO) [89], Brooks and Motley's Poisson model (BMP) [40] and Yamada's S-Shaped growth model (YAM) [261]. 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. [242, 148, 139] for some examples.

Our main goal is to answer the question:

Is there a comparatively better approach for cross-release prediction of fault counts on fault data from large and real world software projects?

To answer it we have identified a number of more detailed research questions listed in Section 3.4. By applying the model creation approaches described above and by answering the research questions the chapter makes the following contributions:

1. Quantitative and qualitative assessment of the generalizability and real-world 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 3.2, we present the background for this study. Section 3.3 elaborates on the data collection procedure. Section 3.4 describes the research questions, while Section 3.6 provides a brief introduction to the techniques used in the study. Section 3.5 describes the different evaluation measures used in the study while Section 3.7 covers the application of different techniques and the corresponding evaluation. The validity evaluation is presented in Section 3.8, while discussion and conclusions are presented in Section 3.9.

## 3.2 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* [147]. 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 [53]. 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. The result is that the prediction problem is seen as being largely unsolvable and NP-hard [53, 224]. Due to a large number of studies covering software quality modeling (for both classifying fault-proneness and predicting software faults), the below references are more representative than exhaustive.

Gao and Khoshgoftaar [87] 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. [263] 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. [143] 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 [187], 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 module as fault-prone) on data sets from two commercial systems.

In [35], optimized set reduction classifications (that generates logical expressions representing patterns in the data) which 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 [137] which supports using logistic regression for software quality classification. Also the study by Denaro et al. [67] used logistic regression to successfully classify faults across homogeneous applications. Basili et al. [26] verified that most of Chidamber and Kemerers object-oriented metrics are useful quality indicators for fault-prone classes. Ohlsson et al. [198] investigated the use of metrics for release $n$ to identify the most fault-prone modules in release $n+1$. Later, in [199], principal component analysis and discriminant analysis was used to rank the software modules 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. [138] 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. [38] proposed multivariate adaptive regression splines (MARS) to classify object-oriented (OO) 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 [183], the authors show that static code attributes like McCabe's and Halstead's are valid attributes for fault prediction. It was further shown that naive Bayes outperformed the decision tree learning methods.

As discussed briefly in Section 3.1, the use of regression analysis might not be the best approach for software fault prediction. Therefore, 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* [47]. These characteristics are also common to data collected from a typical software development process. Karunanithi et al. published several studies [126, 127, 128, 129, 130] using neural network architectures for software reliability growth modeling. Other examples of studies reporting encouraging results include [5, 17, 70, 98, 99, 107, 134, 135, 136, 148, 228, 238, 239, 240, 241]. While, finally, Cai et al. [48] 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. [90] 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. [147] 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.

Gyimothy et al. [100] 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. [163] 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 [49, 50, 230, 249], and support vector machines, as in [242], 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 [61, 200, 59, 266, 8, 6, 7]. 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 [139, 140].

There are also studies that use a combination of techniques, e.g. [242], where genetic algorithms are used to determine an optimal neural network architecture and in [193], where principal component analysis is used to enhance the performance of neural networks.

As mentioned in Section 3.1, very few studies have looked at cross-release predictions of fault data on a large scale. Ostrand and Weyuker [201] 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 [202] 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.

## 3.3 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

[2]http://tomcat.apache.org/
[3]http://www.openbsd.org

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 an 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 corrected before the final builds.

The second and third fault count data sets (IND02, 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 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 $2,000$.

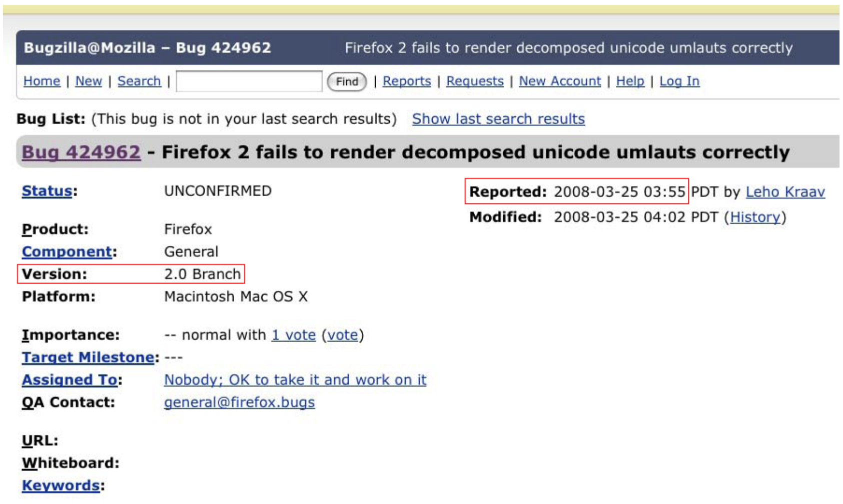
The last data set, IND04, comes from a large mobile hydraulics company specializing in engineered hydraulic, electric and electronic systems. The fault count data set comes from one of their products, a graphical user interface integrated development environment, which is a part of a family of products 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]http://www.mozilla.com/

### 3.3.1 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 3.1 shows a bug report for Mozilla Firefox.


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

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

**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 |
| **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:** | |

**Reported:** 2008-03-25 03:55 PDT by Leho Kraav
**Modified:** 2008-03-25 04:02 PDT (History)

Figure 3.1: A sample bug report.

For OSStom and OSSmoz, we recorded the data from the 'Reported' and 'Version' fields as shown in the Figure 3.1. 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 our data collection because the core development team examines each bug report and decides upon a course to follow [164]. 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 IND01–04. Table 3.1 show more details regarding the data collected from the open source and industry software projects, respectively. 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

Table 3.1: 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 | Length 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<br>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<br>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<br>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<br>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<br>Test on 5.10 | 38 | 11 |
| IND03 | 5.09, 5.10 (Sept. 2005–Dec. 2007) | Train on 5.09<br>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<br>Test on 3.2 | 60 | 16 |

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.

## 3.4 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?

## 3.5 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. [223]. 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 [195].

However, quantitative evaluations of predictive accuracy and bias are not the only important aspects for real world 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.

### 3.5.1 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 [234, 180]. 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) and 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 [146, 177]. We further test a particular model's bias which gives an indication of whether the model is prone to overestimation or underestimation [177]. To measure a particular model's bias, we examine the distribution of residuals to compare models as suggested in [149, 205]. We also formally test for significant differences between competing prediction systems as recommended in e.g. [223]. In the following we describe the evaluation measures in more detail.

**Kolmogorov-Smirnov (K-S) test.** The K-S test is a distribution-free test for measuring general differences in two populations. The statistic $J$ for the two-sample two-sided K-S test is given by,

$$J = \frac{mn}{d} \max_{-\infty < t < +\infty} \{ \mid F_m(t) - G_n(t) \mid \} \tag{3.1}$$

where $F_m(t)$ and $G_n(t)$ are the empirical distribution functions for the two samples respectively, $m$ and $n$ are the two sample sizes and $d$ is the greatest common divisor of $m$ and $n$. In our case, the two samples were, (i) the training part of the actual fault count data and (ii) the actual predictions from the technique under test. For a detailed description of the test, see [109].

**Prequential likelihood ratio (PLR).** PLR is used to investigate the relative plausibility of the predictions from two models [3]. 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 $T_{j+1}, T_{j+2}, \ldots, T_{j+n}$,

$$PL_n = \prod_{i=j+1}^{j+n} \hat{f}_i(t_i) \tag{3.2}$$

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 [39]:

$$\text{PLR}_i^{AB} = \prod_{j=s}^{j=i} \frac{\hat{f}_j^A(t_j)}{\hat{f}_j^B(t_j)} \tag{3.3}$$

In our case, we select the actual time distribution of 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 trust worthy predictions. For further details on PLR, see [39].

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

$$\text{AAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{3.4}$$

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,

$$\text{ARE} = \frac{1}{n}\sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{|y_i|} \tag{3.5}$$

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 $l$, 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 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 [71].

**Distribution of residuals.** To measure a particular model bias, we examine the distribution of residuals to compare models [149, 223]. It has the convenience of applying significance tests and visualizing differences in absolute residuals of competing models using box plots.

### 3.5.2 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 [90]. One or more of these requirements might influence model selection. We use the following qualitative criteria [90, 175, 173, 45], 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?

## 3.6 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 [227] (for running GP), Weka software version 3.4.13 [257] (for running ANN, SVM and LR) and SMERFS3 version 2 [79] (for running GO, YAM and BMP).

### 3.6.1 Genetic programming (GP)

GP is an evolutionary computation technique and is an extension of genetic algorithms [157]. The population structures (individuals) in GP are not fixed length character strings but programs that, when executed, are the candidate solutions to the problem. For the symbolic regression application of GP, programs are expressed 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 worth of an individual GP program in solving the problem is assessed using a fitness evaluation. The fitness evaluation of a particular individual in this case is determined by the correctness of the output produced for all of the fitness cases [21]. 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 [46]. 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. Crossover takes place between two parent trees with swapping branches at randomly chosen nodes, while in tree mutation a random node within the parent tree is substituted with a new random tree created with the available terminals and functions. Reproduction causes a proportion of trees to be copied to the next generation without any genetic operation [227].

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 3.2. 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 [227]. 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 [227].

Table 3.2: 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 |

### 3.6.2 Artificial neural networks (ANN)

The development of artificial neural networks is inspired by the interconnections of biological neurons [219]. 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 [219]. Each neuron computes the weighted sum 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 [257]. 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.

### 3.6.3 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 [219]. For regression, the basic idea is to discard the deviations up to a user specified parameter $\in$ [257]. 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 [97].

### 3.6.4 Linear regression (LR)

The linear regression used in the study performs a standard least-squares linear regression [122]. 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.

### 3.6.5 Traditional software reliability growth models

As discussed in Section 3.1, 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 [89, 261, 40].

The Goel-Okumoto non-homogeneous Poisson process model (GO) [89] is given by,

$$m(t) = a[1 - e^{-bt}] \tag{3.6}$$

while Yamada's S-shaped growth model (YAM) [261] is also a non-homogeneous Poisson process model given by,

$$m(t) = a(1 - (1 + bt)e^{-bt}) \tag{3.7}$$

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 [237]:

$$P[X(t) = n] = \frac{[m(t)]^n e^{-m(t)}}{n!} \tag{3.8}$$

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) [40]. 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,

$$P[X = n_i] = \begin{cases} \frac{(N_i \phi_i)^{n_i} e^{-N_i \phi_i}}{n_i!} \\ \phi_i = 1 - (1 - \phi)^{t_i} \end{cases} \tag{3.9}$$

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

## 3.7 Experiment and results

We have collected data from a total of 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 have 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 [31]. 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.

### 3.7.1 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. The null hypothesis here is that the predicted and the actual fault count data have the same probability distribution [109] i.e.,

$$H_0 : [F(t) = G(t), \text{for every } t] \tag{3.10}$$

At significance level $\alpha = 0.05$, if the K-S statistic $J$ is greater than or equal to the critical value $J_\alpha$, the null hypothesis (Eq. 3.10) is rejected in favor of the alternate hypothesis, i.e. that the two samples do *not* have the same probability distribution.

Table 3.3 shows the results of applying K-S test statistic for each technique for every data set. The (–) in the Table 3.3 indicates that the algorithm was not able to converge for the particular data set. The instances where the K-S statistic $J$ is less than the critical value $J_\alpha$ are shown in bold in Table 3.3. It is evident from Table 3.3 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 3.4 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

Table 3.3: Results of applying Kolmogorov-Smirnov test. The bold values indicate $J<J_\alpha$, (–) indicates lack of model convergence, $J_\alpha$ is the critical $J$ value at $\alpha$=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 |

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 3.4. 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 3.4: 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.

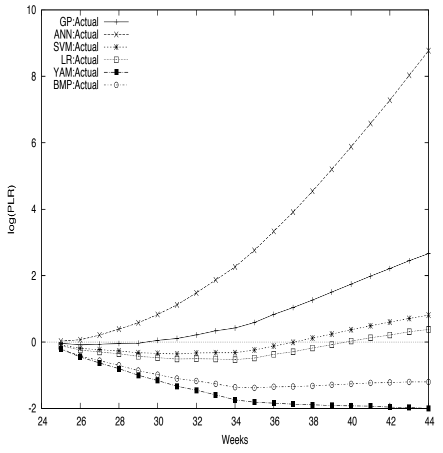
### 3.7.2 Evaluation of predictive accuracy

Table 3.5 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 3.5, 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 3.5. 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.

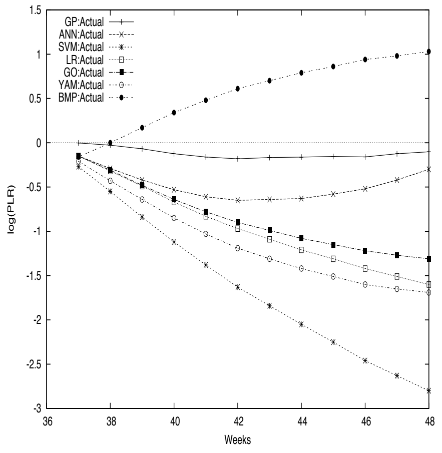
Table 3.5: $log(\text{PLR})$ values for one-step-ahead predictions. The values shown are the final log result of the running product of ratio of the successive on-step ahead predictions of actual fault count and other models' predictions. Values closer to 0 are better.

| | 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 |

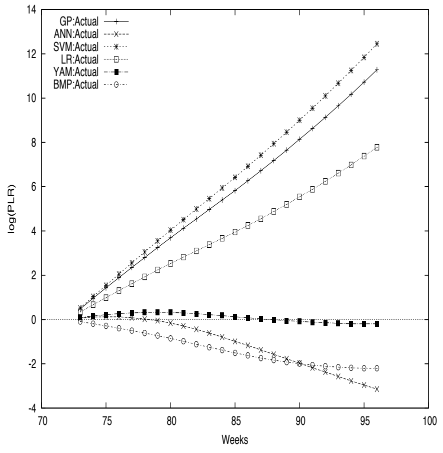
Figure 3.7.2 depicts the PLR analysis for all the data sets which shows the pair-wise comparisons of each technique with the actual weekly/monthly fault count data which has been chosen as the reference model (indicated as a dotted straight line in the plots of Figure 3.7.2). We see that for OSStom (Figure 3.2a), the prediction curves for LR and SVM are closer to the reference in comparison with other curves. For OSSbsd (Figure 3.2b), 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 3.2d, 3.2e and 3.2g). However, for OSSmoz (Figure 3.2c), YAM is better at following the reference compared to any other curve, while for IND03 (Figure 3.2f), 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 of the seven data sets.



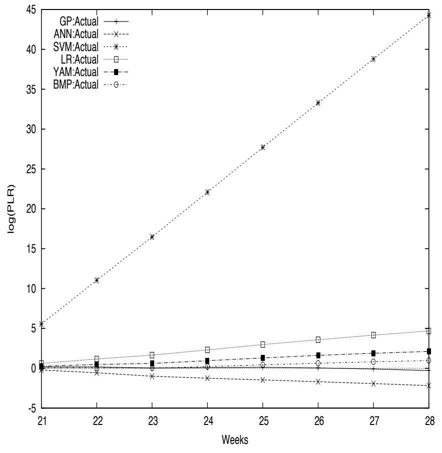
(a) OSStom.



(b) OSSbsd.

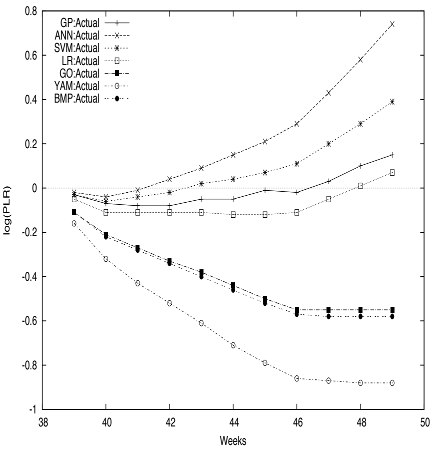


(c) OSSmoz.

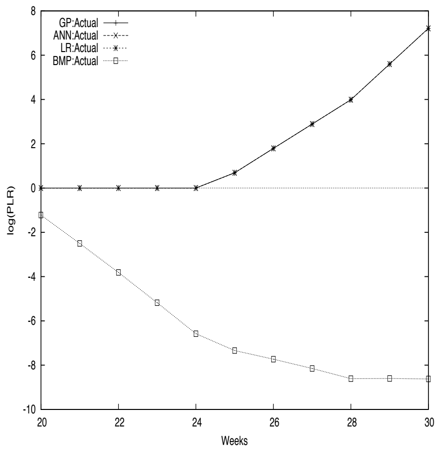


(d) IND01.

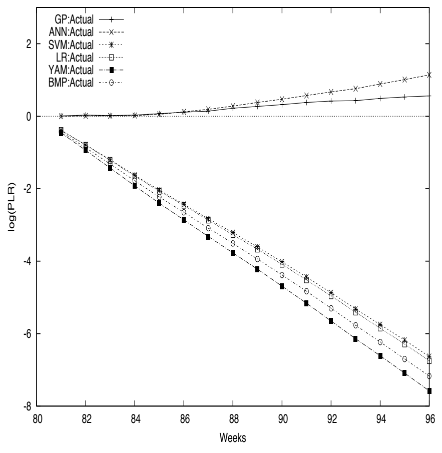
Figure 3.2: $log(\text{PLR})$ plots for the data sets OSStom, OSSbsd, OSSmoz, IND01, IND02, IND03 and IND04. Continuing on to the next page.



(e) IND02.



(f) IND03.



(g) IND04.

Figure 3.2: Continuing from the previous page; $log(\text{PLR})$ plots for the data sets OSStom, OSSbsd, OSSmoz, IND01, IND02, IND03 and IND04.

Table 3.6 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 3.6: 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 3.7 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 3.8 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 shows that GP is generally a better approach for variable-term predictability.

As with AAE, ARE samples from different methods also did not satisfy the normality 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.

Table 3.7: 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 3.8: 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 |

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 3.9 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.

Table 3.9: 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 |

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 3.10.

Table 3.10: 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 |

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 to other techniques, the summary statistics (Tables 3.7 and 3.9) 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.

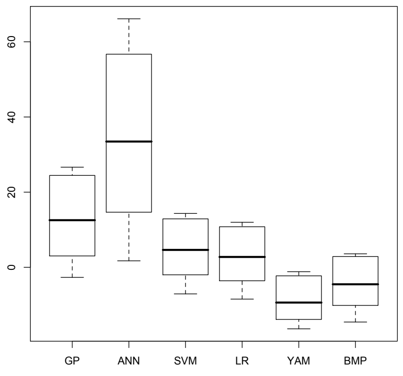
### 3.7.3 Evaluation of model bias

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 Figure 3.7.3. For OSS-bsd (Figure 3.3b) and IND04 (Figure 3.3g), the box plot of GP show two important characteristics:

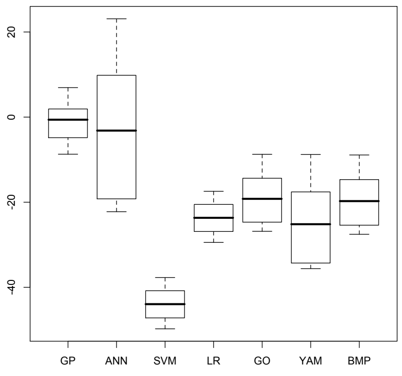
1. Smaller or equivalent length of the box plot as compared with other box plots.
2. Presence of majority of the residuals close to 0 as compared with other box plots.

For IND03 (Figure 3.3f), the length of the box plot and its proximity close to 0 appear to be similar for GP, ANN and LR. For OSStom (Figure 3.3a), SVM and LR are better placed than the rest of the techniques while for OSSmoz (Figure 3.3c), YAM 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 3.3e) where GP and LR show a good trade-off between length and actual position of the box plot.

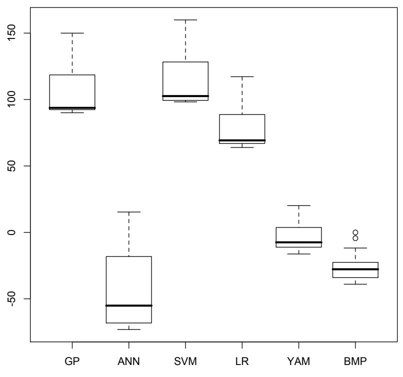
Since the box plots of the residuals were skewed, we resorted to using the non-parametric Kruskal-Wallis test to examine if there is a statistical difference between the residuals for all the data sets and to confirm the trend observed from the box plots. The results of the application of the Kruskal-Wallis test appear in Table 3.11. For each of the data sets, the Kruskal-Wallis statistic $h$ is greater than the critical value $\chi^2_{0.05}$. Therefore, we had sufficient evidence to reject the null hypothesis that the residuals for different techniques within a project were similar.



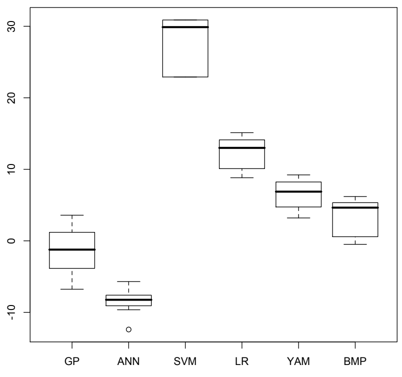
(a) Residuals for OSStom.



(b) Residuals for OSSbsd.

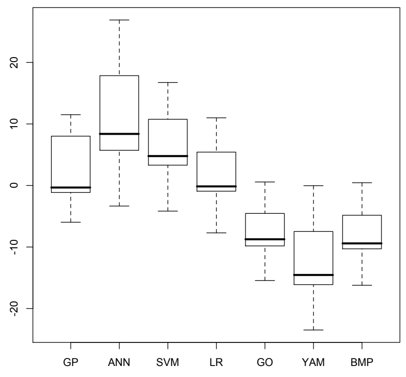


(c) Residuals for OSSmoz.

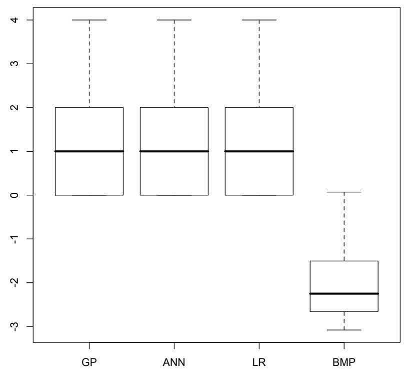


(d) Residuals for IND01.

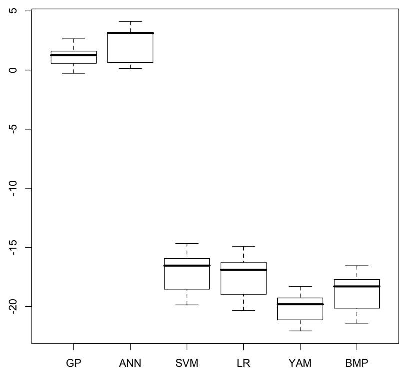
Figure 3.3: Charts showing box plots of residuals for the seven data sets. Continuing on to the next page.



(e) Residuals for IND02.



(f) Residuals for IND03.



(g) Residuals for IND04.

Figure 3.3: Continuing from the previous page; Charts showing box plots of residuals for the seven data sets.

Table 3.11: Kruskal-Wallis statistic $h$ for different data sets for testing difference in residuals. $\nu$ 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 3.12. 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 3.12.)

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

Table 3.12: $p$-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 |

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 techniques.

### 3.7.4 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 [173]. 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) [79, 194], 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 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* [90]. In our case, with one independent variable (week number) and one dependent variable (count of faults), typical GP solutions are of the form below:

$$times(minus(sin(minus(cos(x),x)),minus(log(cos(log(sin(log(x))))),sin(x))),log(x)) \tag{3.11}$$

where $x$ is the independent variable and *minus*, *times*, *sin*, *cos*, *log* represents the function set (as outlined in Table 3.2).

**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 (as in Eq.3.11), a phenomenon known as bloating. Although there are different ways to control this (see e.g. [170]), 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 [90]. The machine learning and evolutionary models certainly scores high in this respect in comparison with traditional methods.

## 3.8 Validity evaluation

There can be different threats to the validity of the empirical results [258]. 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 [121]; however, facing some criticism lately [112]. 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 3.6.1) is similar to the quantitative evaluation measures for comparing different techniques (Subsection 3.5.1). This is, however, not the case with this study since the GP fitness function differs from the quantitative evaluation measures and also 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 could 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. [259]) 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 can not 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 real-life fault data from multiple software releases. 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.

## 3.9 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 [175] 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 its 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 [74]. 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 parsimonious factor takes into account that the model with the smallest number of parameters is usually the best.

4. Potentially poor generalizability outside the range of the training data [155]

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 [156]. 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 3.13.

Table 3.13: 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 3.13 are given as to summarize the detailed evaluation done in the study based on several measures (Section 3.5). 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 3.7.4. Table 3.13 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 3.13 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 4 is a systematic literature review investigating the extent of application of symbolic regression using genetic programming within software engineering predictive modeling.

Chapter 4

# Genetic programming versus other techniques for software engineering predictive modeling: A systematic review

## 4.1 Introduction

Genetic programming (GP) [157] 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* [208]. 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 [156].

This paper 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. [160], software size, e.g. [169], software development cost/effort, e.g. [120], maintenance task effort, e.g. [117], correction cost, e.g. [65], software fault, e.g. [236], and software release timing, e.g. [70]. 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 [52], while a systematic review of software development cost estimation studies is provided by [120]. The current study differs from these systematic reviews in several ways. Firstly, the studies of [52] and [120] 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, [52] and [120] 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 4.2.1). Thirdly, one additional concern, which makes this study different from studies of [52] and [120], is that it also assesses the evidence of comparative analysis of symbolic regression with other competing models.

A paper by by Crespo et al.[62] which 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 of us carrying out this systematic review is the general growing interest in search-based approaches to solve software engineering problems [105]. In this regards, 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 rest of the text below, wherever we refer to GP, we mean the symbolic regression application of it.

This paper is organized as follows: Section 4.2 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 4.3, while Section 4.4 discusses the results and future work. Validity threats and conclusions appear in Section 4.5 and Section 4.6, respectively.

## 4.2 Method

This section describes our review protocol, which is a multi-step process following the guidelines outlined in [150].

### 4.2.1 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 [204]. The *population* is 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.

### 4.2.2 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 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. [159] and Alander's bibliography of genetic programming [13] 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 4.2.1):

- **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:

- 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. [72, 45, 59]. All of the known papers were found using multiple digital libraries.

We additionally scanned the online GP bibliography maintained by Langdon et al. [159] by using the search-term *symbolic regression*. We also searched an online data base of software cost and effort estimation called BESTweb [118], 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 [159, 13]. 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.

### 4.2.3 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 [150]. We excluded studies that:

1. Do not relate to software engineering or software development, e.g. [16].

2. Do not relate to prediction/estimation of software cost/effort/size, faults, quality, maintenance, correction cost and release timing, e.g.[4].

3. Report performance of a particular technique/algorithmic improvement without being applied to software engineering, e.g. [19].

4. Do not relate to symbolic regression (or any of its variants) using genetic programming, e.g. [226].

5. Do not include a comparison group, e.g. [139].
6. Use genetic programming only for feature selection prior to using some other technique, e.g. [214].
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. [59] but exclude two of theirs conference papers [61, 200].

Table 4.1 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 4.1: 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 [159] | 342 (342) |
| BESTweb [118] | 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.

### 4.2.4 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 [150]. We did not rank the studies according to an overall quality score but used a simple 'yes' or 'no' scale [76]. Table A.1, in Appendix A, shows the application of the study quality assessment criteria where a ($\surd$) indicates 'yes' and ($\times$) indicates 'no'. Further a ($\sim\surd$) shows that we were not sure as not enough information was provided but our inclination is towards 'yes' based on reading full text. A (~×) 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 [150, 152]:

- 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.

## 4.3 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 4.2 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 4.1 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 little surprising; considering that the proponents of symbolic regression application of GP have highlighted several advantages of using it [209].

In the right division of Figure 4.1, 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.

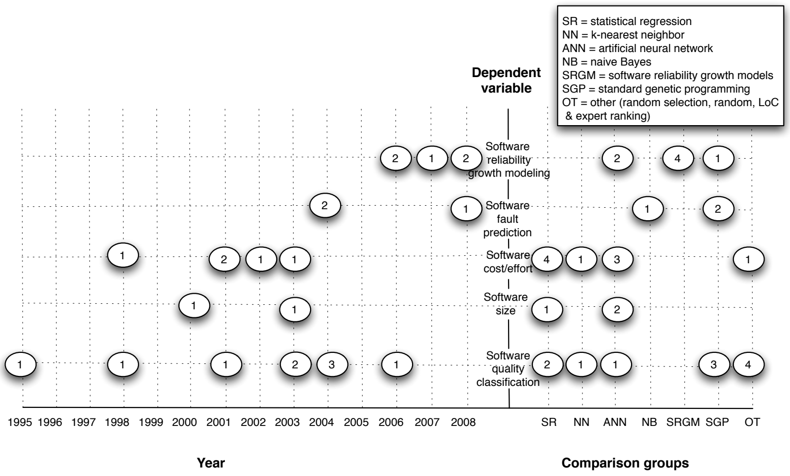


Figure 4.1: Distribution of primary studies over range of applied comparison groups and time period.

Table 4.2: Distribution of primary studies per predicted/estimated attribute.

| Domain | Author(s) | Year | Ref. |
|---|---|---|---|
| SW quality classification (37.50%) | Robinson et al. | 1995 | [217] |
| | Evett et al. | 1998 | [78] |
| | Khoshgoftaar et al. | 2003 | [145] |
| | Liu et al. | 2001 | [168] |
| | Khoshgoftaar et al. | 2004 | [140] |
| | Khoshgoftaar et al. | 2004 | [141] |
| | Liu et al. | 2004 | [166] |
| | Reformat et al. | 2003 | [215] |
| | Liu et al. | 2006 | [167] |
| SW CES estimation (29.17%) | Dolado et al. | 1998 | [73] |
| | Dolado | 2000 | [71] |
| | Regolin et al. | 2003 | [216] |
| | Dolado | 2001 | [72] |
| | Burgess et al. | 2001 | [45] |
| | Shan et al. | 2002 | [222] |
| | Lefley et al. | 2003 | [161] |
| SW fault prediction and reliability growth (33.33%) | Kaminsky et al. | 2004 | [123] |
| | Kaminsky et al. | 2004 | [124] |
| | Tsakonas et al. | 2008 | [247] |
| | Zhang et al. | 2006 | [266] |
| | Zhang et al. | 2008 | [267] |
| | Afzal et al. | 2008 | [6] |
| | Costa et al. | 2007 | [59] |
| | Costa et al. | 2006 | [60] |

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

### 4.3.1 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 over-lapped 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 distinct contribution and thus worthy of inclusion as primary studies for this 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 module as being either fault-prone (*fp*) or not fault-prone (*nfp*). A fault-prone module 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 modules. 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 modules with known classification as fault-prone or not fault-prone. The generated model is then tested to predict the risk-based class membership of modules 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 [145]. This means that the GP models can potentially show the significant software metrics affecting the quality of modules. 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 [217], the authors use GP to identify fault-prone software modules. A software module 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 [78] the authors describe a GP-based system for targeting software modules for reliability enhancement. This study not only predicted the number of faults but also ranked-order the software modules. The motivation was to assist the project managers in deciding to select those software modules that 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 [217] 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 modules 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 modules for reliability enhancement. With cut-off percentile values of 75%, 80%, 85% and 90% for module ordering, GP model performance was consistently superior to random ordering of modules 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. [140], 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%) and (*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 modules closer to the actual ranking on two of the three data splits. The results were not much different in an extension of this study [141], where in an additional case study of a legacy telecommunication system with 28 independent variables, GP outperformed the module ranking based on LOC.

Another study by Khoshgoftaar et al. [145] 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 error (a *fp* module misclassified as *nfp*) more than Type-I error (a *nfp* module misclassified as *fp*). This was done by normalizing the cost of Type-II error with respect to the cost of Type-I error. Data was collected from two embedded systems applications, which consisted of five different metrics for different modules. 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 [168] in which a single objective fitness function was used which 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 [166].

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 [215], 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 best splitting of attribute domains for the decision tree and to get a 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 [167], 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 5 *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 4.3 and 4.4[1] show the relevant summary data extracted to answer the research question from each of the primary studies within software quality classification.

### 4.3.2 Software cost/effort/size (CES) estimation

In line with what Jørgensen and Shepperd suggest in [120], we will use the term "cost" and "effort" interchangeably since effort is a typical cost driver in software development.

[1]The data sets in Table 4.4 are taken at a coarser level, e.g. ISBSG data ([113]) of multiple projects is 1 data set.

Table 4.3: Summary data for primary studies on GP application for software quality classification. (?) indicates absence of information and (~) indicates indifferent results.

| Article | Dependent variable | Fitness function | Comparison group | Evaluation measures | GP better? |
|---|---|---|---|---|---|
| [217] | Fault proneness based on number of faults | Minimization of root mean square | Neural networks, $k$-nearest neighbor, linear regression | Accuracy & coverage | ~ |
| [78] | Fault proneness based on number of faults | Minimization of absolute errors as well as maximization of best percentage of actual faults averaged over the percentile level of interest | Random rank ordering | Ranking on the basis of faults in different percentile ranges | √ |
| [145] | Fault proneness based on number of faults | Minimization of average cost of misclassification and minimization of tree size | Standard GP | Type I, Type II and overall error rates | √ |
| [168] | Fault proneness based on number of faults | Minimization of the average cost of misclassification | Logistic regression | Type I, Type II and overall error rates | √ |
| [140] | Number of faults for each software module | Maximization of the best percentage of actual faults averaged over the percentiles level of interest and controlling the tree size | Ranking based on lines of code | Number of faults accounted by different cut-off percentiles | √ |
| [141] | Number of faults for each software module | Maximization of the best percentage of actual faults averaged over the percentiles level of interest and controlling the tree size | Ranking based on lines of code | Number of faults accounted by different cut-off percentiles | √ |
| [166] | Fault proneness based on number of faults | Minimization of expected cost of misclassification and controlling the tree size | Standard GP | Number of over-fitting models and Type I, Type II error rates | √ |
| [215] | Ranking of object's quality | (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 | ~ |
| [167] | Fault proneness based on number of faults | Minimization of the expected cost of misclassification and controlling the tree size | Standard GP | Type I and Type II error rates | √ |

Table 4.4: 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 |
|---|---|---|---|---|
| [217] | 1 | 103 records for training and 60 records for testing | ? | Private |
| [78] | 2 | $\frac{2}{3}$ modules for training and the rest for testing | I | Private |
| [145] | 1 | Approximately $\frac{2}{3}$ for training and the rest for testing | I | Private |
| [168] | 1 | Approximately $\frac{2}{3}$ for training and the rest for testing and random subset selection | I | Private |
| [140] | 1 | $\frac{2}{3}$ for training and the rest for testing, three splits | I | Private |
| [141] | 2 | $\frac{2}{3}$ for training and the rest for testing, three splits | I | Private |
| [166] | 1 | Training on release 1 data set, testing on release 2,3,4 data sets | I | Private |
| [215] | 1 | 10 fold cross-validation | I | Private |
| [167] | 7 | 1 training data set, 5 validation data sets and 1 testing data set | I | Private |

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. [62], 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 [73], 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 $\mathrm{pred}(0.25)$ 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 [71], 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 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.[216] used a similar approach of estimating LOC from function points and number of components (NOC) metric. Using GP and NN, the prediction models using function points did not satisfy the criteria MMRE $\leq 0.25$ and pred$(0.25) \geq 0.75$. However, the prediction models for estimating lines of code from NOC metric were acceptable from both the NN and the GP point of view.

In [72], 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. [45] extends the previous study from [72] by using 9 independent variables to predict the dependent variable of effort measured in person hours. Using the Desharnais 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 [222] the authors used grammar-guided GP on 423 projects from release 7 of the ISBSG (The International Software Benchmarking Standards Group Limited [113]) 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 [161] 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 4.5 and 4.6[2] present the relevant summary data extracted to answer the research question from each of the primary studies within software CES estimation.

### 4.3.3 Software fault prediction and reliability growth

Apart from studies on software quality classification (Section 4.3.1), where the program modules 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 [123] 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 [12], Halstead's equation [101], Lipow's equation [165], Gaffney's equation [86] and Compton's equation [57] 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 that by including explicit knowledge in the GP solutions, the fitness values for the GP solutions increased.

In another study, [124], 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

[2]The data sets in Table 4.6 are taken at a coarser level, e.g. ISBSG data ([113]) of multiple projects is 1 data set.

Table 4.5: Summary data for primary studies on GP application for software CES estimation. (~) indicates indifferent results.

| Article | Dependent variable | Fitness function | Comparison group | Evaluation measures | GP better? |
|---|---|---|---|---|---|
| [73] | Software effort | Mean squared error | Neural networks & linear regression | pred(0.25) [3] and MMRE [4] | ~ |
| [71] | Software size | Mean squared error | Neural networks & multiple linear regression | pred(0.25) and MMRE | ~ |
| [216] | Software size | MMRE | Neural networks | pred(0.25) and MMRE | ~ |
| [72] | Software cost | Mean square error | Linear, logarithmic, inverse quadratic, cubic, power, exponential, growth and logistic regression | pred(0.25) and MMRE | ~ |
| [45] | Software effort | MMRE | neural networks | MMRE, AMSE [5], pred(0.25), BMMRE [6] | ~ |
| [222] | Software effort | Mean square error | Linear regression, log-log regression | $R$-squared [7], MMRE, pred(0.25) and pred(0.5) | ~ |
| [161] | Software effort | ? | 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 | ~ |

[3]Prediction at level 0.25
[4]Mean Magnitude of Relative Error
[5]Adjusted Mean Magnitude of Relative Error
[6]Balanced Mean Magnitude of Relative Error
[7]Coefficient of multiple determination

Table 4.6: 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 |
|---|---|---|---|---|
| [73] | 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 |
| [71] | 6 | Train a model on 60 to 67 % of the data points and test in 40 to 37% | A | Public |
| [216] | 2 | Train on $\frac{2}{3}$ and test on $\frac{1}{3}$ | I & A | Public |
| [72] | 12 | Training and testing on all data points | I & A | Public |
| [45] | 1 | Training on 63 projects, testing on 18 projects | I | Public |
| [222] | 1 | Random division of 50% in training set and 50% in testing set | I | Public |
| [161] | 1 | 149 projects in the training set and 15 projects in the testing set | I | Public |

that the average fitness values for the equalized data set were better than for the original data set.

In [247], 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 modules found to all known faulty modules) and false alarm rate, PF (the ratio of number of non-faulty modules misclassified as faulty to all known non-faulty modules). 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 naive 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. [266] 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 [267] 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 [59], 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), $\sqrt{\sum_{i=1}^{m} (x_i - x_i^d)^2 D_i m}$ 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 [200], 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 $\lambda$ individuals, which competed with their parents in the selection of $\mu$ 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 4.7 and Table 4.8[8] shows the relevant summary data extracted to answer the research question from each of the primary studies within software fault prediction and reliability growth.

## 4.4 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 [217], or the proposed technique used GP for initial investigative purposes only, without being definitive in the judgement of GP's efficacy, as in [215] (these two studies are indicated by the sign ~ in Table 4.3).

The other seven studies were co-authored by similar authors to a large extent and the data sets also over-lapped between studies but these studies contributed in introducing different variations of the GP fitness function and also used different comparison groups.

[8]The data sets in Table 4.8 are taken at a coarser level, e.g. ISBSG data ([113]) of multiple projects is 1 data set.

Table 4.7: Summary data for primary studies on GP application for software fault prediction and reliability growth. (?) indicates absence of information and (~) indicates indifferent results.

| Article | Dependent variable | Fitness function | Comparison group | Evaluation measures | GP better? |
|---|---|---|---|---|---|
| [123] | Software fault prediction | $1 - \text{standard error}$ | Standard GP | Fitness values | √ |
| [124] | Software fault prediction | $1 + e^{(7*(1-n-k)/(n-1)*Se^2/Sy^2)}$ | Standard GP | Fitness values | √ |
| [247] | Software fault prediction | $\frac{tp}{(tp+fn)} * \frac{tn}{(tn+fp)}$ | Naive Bayes | PD & PF | ~ |
| [266] | Software reliability | ? | 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 | √ |
| [267] | Software reliability | ? | Traditional software reliability growth models | Mean time between failures and next mean time to failure | √ |
| [6] | Software reliability | $\sum_{i=1}^{n} \mid e_i - e_i' \mid$ | Traditional software reliability growth models | Prequential likelihood ratio, AMSE, Braun statistic, Kolmogorov-Smirnov test and distribution of residuals | √ |
| [59] | Software reliability | WRMSE [9] | Traditional software reliability growth models and ANN | Maximum deviation, average bias, average error, prediction error, correlation coefficient, Kolmogorov-Smirnov | √ |
| [200] | Software reliability | RMSE [10] | Standard GP | Maximum deviation, average bias, average error, prediction error and correlation coefficient | √ |

[9]Weighted root mean square error

[10]Root mean square error

Table 4.8: 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 |
|---|---|---|---|---|
| [123] | 1 | ? | I | Public |
| [124] | 1 | ? | I | Public |
| [247] | 1 | 10-fold cross-validation | I | Public |
| [266] | 1 | ? | I | Private |
| [267] | 1 | ? | I | Private |
| [6] | 3 | First $\frac{2}{3}$ of the data set for training and the rest for testing | I | Private |
| [59] | 2 | First $\frac{2}{3}$ of the data set for training and the rest for testing | I | Public & Private |
| [200] | 1 | First $\frac{2}{3}$ of the data set for training and the rest for testing | I | Public |

cThese 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 modules 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 [145, 166]. 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.

We also observe from Table 4.3 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.

There are many different techniques that have been applied by researchers to software quality classification, see e.g. [163]. 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 4.4 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 4.4 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 4.5). The sign ~ in the last column of Table 4.5 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 [85, 223]. Another aspect which differed between the studies was the sampling strategies used for training and testing sets (Column 3, Table 4.6). 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 4.3.2, Crespo et al. [62] 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, naive Bayes and traditional software reliability growth models (this is evident from the last column in Table 4.7). However, as Table 4.8 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 [123]. Table 4.7 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 4.9 (an approach similar to the one used in [176]) that reflects the indifference between GP and other approaches.

What is evident from these studies is the following:

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 [156]. 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 [71] provides the following equation generated by GP:

   $LOC = 50.7 + 1.501 * \textit{data elements} + \textit{data elements} * \textit{relations} - 0.5581 * \textit{relations}$

   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 4.3, 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 researchers.

Table 4.9: Summary of the studies showing inconclusive results in using GP.

| Article | Quotation |
|---|---|
| [217] | While generally not as good as the results obtained from other methods, the GP results are reasonably accurate but low on coverage. |
| [215] | 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%. |
| [73] | However, from the point of view of making good predictions, no technique has been proved to be clearly superior.<br>. . . 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. |
| [71] | 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.<br>. . . 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. |
| [216] | We cannot conclude GP is a better technique than NN.<br>. . . GP and ANN are valid and promising approaches to size estimation.<br>. . . However, GP presents additional advantages with respect to NN. The main advantage of using GP is the easy interpretation of result. |
| [72] | 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. |
| [45] | 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 . . . |
| [222] | 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. |
| [161] | 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. |
| [247] | 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. |

## 4.5 Validity threats

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 [176]. Also, like any other systematic review, this one too is limited to making use of information given in the primary studies [152]. 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 4.2.2) have given us a complete set. Our study selection procedure (Section 4.2.3) 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.

## 4.6 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 modules 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. This is because 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, naive Bayes and traditional software reliability growth models. Although four out of these eight studies lacked in some of the quality instruments used in Table A.1 (Appendix A); still three out of the remaining four studies reported results in support of GP. We therefore concluded that the 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 wherever 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 [259]).

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. [152] 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 [25]).

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 [191].

7. Report the results even if there is no statistical difference between the quality of the solutions produced by different methods [25].

# Chapter 5

# A systematic review of search-based testing for non-functional system properties

## 5.1 Introduction

Search-based software engineering (SBSE) is the application of optimization techniques in solving software engineering problems [105, 103]. The applicability of optimization techniques in solving software engineering problems is suitable as these problems frequently encounter competing constraints and require near optimal solutions. Search-based software testing (SBST) research has attracted much attention in recent years as part of a general interest in SBSE approaches. The growing interest in SBST can be attributed to the fact that generation of software tests is generally considered as an undecidable problem, primarily due to the many possible combinations of a program's input [182]. All approaches to SBST are based on satisfaction of a certain test adequacy criterion represented by a fitness function [103]. McMinn [182] has written a comprehensive survey on search-based software test data generation. The survey shows the application of metaheuristics in white-box, black-box and grey-box testing. Within the domain of non-functional testing, the survey indicates the application of metaheuristic search techniques for checking the best-case and worst-case execution times (BCET, WCET) of real-time systems. McMinn highlights possible directions of future research into non-functional testing, which includes searching for input situations that break memory or storage requirements, automatic detection of memory leaks, stress testing and security testing. Our work extends the survey by McMinn [182] as it analyses actual evidence supporting McMinn's ideas of future directions in search-based testing of non-functional properties. Moreover, we anticipated studies making use of search-based techniques to test non-functional properties not highlighted by McMinn. This work also supports McMinn's survey by finding further evidence into search-based execution time testing. Another review by Mantere and Alander [178] highlights work using evolutionary computation within software engineering, especially software testing. According to the review, genetic algorithms are highly applicable in testing coverage, timings, parameter values, finding calculation tolerances, bottlenecks, problematic input combinations and sequences. This study also extends and supports Mantere and Alander's review in actually finding the evidence in support of proposed future extensions.

Within non-functional search-based software testing (NFSBST) research, it is both important and interesting to know the extent of application of metaheuristic search techniques to non-functional testing, not covered by previous studies. This allows us to identify potential non-functional properties suitable for applying these techniques and provides an overview of existing non-functional properties tested using metaheuristic search techniques. In this paper, after identifying existing non-functional properties, we review each of the properties to determine any constraints and limitations. We also identify the range of different fitness functions used within each non-functional property, since the fitness function is crucial in guiding search into promising areas of solution space and is the differentiating factor between quality of different solutions. The contribution of this review is therefore an exploration of non-functional properties tested using metaheuristic search techniques, identification of constraints and limitations encountered and an analysis of different fitness functions used to test individual non-functional property.

Section 5.2 describes the method of our systematic review that includes the research questions, search strategy, study selection criteria, study quality assessment and data extraction. Sections 5.3 and 5.4 discusses the results, synthesis of findings, areas of future research and validity threats. Conclusions are presented in Section 5.6.

## 5.2 Method

A systematic review is a process of assessment and interpretation of all available research related to a research question or subject of interest [150]. Kitchenham [150] also describes several reasons of undertaking a systematic review, the most common are to synthesize the available research concerning a treatment or technology, identification of topics for further investigation and formulation of a background in positioning new research activities.

This section describes our review protocol, consisting of several steps as outlined in [150].

### 5.2.1 Research questions

In order to examine the evidence of testing non-functional properties using metaheuristic search techniques, we have the following research questions:

- **RQ 1.** In which non-functional testing areas have metaheuristic search techniques been applied?

After having identified these areas, we have three additional research questions applicable in each area:

- *RQ 1.1.* What are the different metaheuristic search techniques used for testing each non-functional property?
- *RQ 1.2.* What are the different fitness functions used for testing each non-functional property?
- *RQ 1.3.* What are the current challenges or limitations in the application of metaheuristic search techniques for testing each non-functional property?

The *population* in this study is the domain of software testing. *Intervention* includes application of metaheuristic search techniques to test different types of non-functional properties. The *comparison intervention* is not applicable in our case as our research questions are not aimed at making a comparison. However, we discuss the comparisons within the scope of each primary study to support our argumentation of obtained results in Section 5.4.

The *outcome* of our interest represents different types of non-functional testing that use metaheuristic search techniques. In terms of *context* and *experimental design*, we do not enforce any restrictions.

### 5.2.2 Generation of search strategy

The search strategy was based on the following steps:

1. *Identification of alternate words and synonyms for terms used in the research questions.* This is done to minimize the effect of differences in terminologies.

2. *Identify common non-functional properties for searching.* We take non-functional properties as to encompass the three aspects of software quality defined in ISO/IEC 9126-1 [114]. These aspects are quality in use, external quality and internal quality. Quality in use refers to software product quality in a specific context of use, while external quality is the quality observable at software execution. Lastly, internal quality is measured against the internal quality requirements.

   Since there are different systems of categorizing non-functional properties, we take guidance from four existing taxonomies to aid our search strategy and to have a representative set of non-functional properties. These are Boehm software quality model (as described in [83]), ISO/IEC 9126-1 [114], IEEE Standard 830-1998 [1] and Donald G. Firesmith's taxonomy [84]. The non-functional properties used for searching are usability, safety, robustness, capacity, integrity, efficiency, reliability, maintainability, testability, flexibility, reusability, portability, interoperability, security, performance, availability and scalability. To cover other potential non-functional properties, we explicitly used the term 'non-functional' in our search strings.

   The non-functional properties obtained from existing taxonomies are restricted to high-level external attributes only for the sole purpose of guiding the search strategy. The different non-functional testing areas that are discussed later in the paper cannot be mapped one to one with these listed non-functional properties. Therefore, while quality of service includes attributes; namely availability and reliability, we have retained the term quality of service for the later part of the paper to better reflect the terms as used by the original authors. Similarly, one can argue execution time to fit under performance, but we stick to the term execution time in the later part of the paper to remain consistent with the terms used by the original authors.

3. *Use of Boolean `OR` to join alternate words and synonyms.*

4. *Use of Boolean `AND` to join major terms.*

We used the following search terms:

- **Population:** testing, software testing, testing software, test data generation, automated testing, automatic testing.

- **Intervention:** evolutionary, heuristic, search-based, metaheuristic, optimization, hill-climbing, simulated annealing, tabu search, genetic algorithms, genetic programming.
- **Outcomes** non-functional, safety, robustness, stress, security, usability, integrity, efficiency, reliability, maintainability, testability, flexibility, reusability, portability, interoperability, performance, availability, scalability

We used a two-phase strategy for searching. In the first phase, we searched electronic databases and performed a manual search of specific conference proceedings and journals. We selected 1996 as the starting year for the search since this year marked the first publication of the application of genetic algorithms to execution time testing [182] (one of the earliest non-functional properties to be tested using metaheuristic search techniques). We searched within the following electronic databases:

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

In the first phase of the search strategy, we piloted the search strings thrice for the year 2007, each time refining them to eliminate irrelevant hits. We found it as a useful activity to pilot the search strings in iterations as it resulted in much refinement of search results. It also helped us to deal with the challenging task of balancing comprehensiveness versus precision of our search. We applied separate search strings for searching within titles, abstracts and keywords. Complete search strings are given in Appendix C.

We manually searched selected journals (J) and conference proceedings (C). These journals and conferences were chosen as they had previously published primary studies relevant to our domain. They include: Real Time Systems Symposium (RTSS) (C), Real Time Systems (RTS) (J), Genetic and Evolutionary Computation Conference (GECCO)[1]—Search-based Software Engineering (SBSE) Track (C), Software Testing, Verification and Reliability (STVR) (J) and Software Quality Journal (SQJ) (J).

We initiated a second phase of search to have a more representative set of primary studies. In this phase, we scanned the reference lists of all the primary studies to identify further papers. We then contacted the researchers who authored most of the

[1]GECCO was not part of ACM until 2005.

papers in a particular non-functional area for additional papers. Moreover, we scanned the personal web pages maintained by these researchers. A total of four researchers were contacted. Figure 5.1 shows our two-phase search strategy.

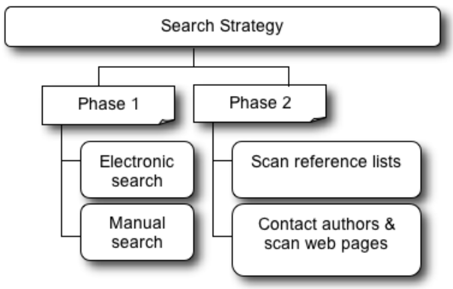


Figure 5.1: The two-phase search strategy.

In order to assess the results of the search process, we compared the results with a small sample of primary studies we already knew about ([253, 243, 36]), to ensure that the search process was able to find the sample (as described in [152]). All the three known papers were found using nine sources, namely (IEEE Xplore, Compendex, Web of Science, ACM Digital Library, Real-Time Systems Symposium (C), Real-Time Systems (J), GECCO SBSE track (C), Software Testing, Verification and Reliability (J), Software Quality Journal (J)).

### 5.2.3 Study selection criteria and procedures for including and excluding primary studies

Metaheuristic search techniques have been applied across different engineering and scientific disciplines. Within software testing, metaheuristic search techniques have found application in different phases, from planning to execution. Therefore, it is imperative that we define comprehensive inclusion/exclusion criteria to select only those primary studies that provide evidence related to the research questions. The following exclusion criteria is applicable in this review, i.e. exclude studies that:

- Do not relate to software engineering/development.
- Do not relate to software testing.

- Do not report application of metaheuristics. (We consider metaheuristics to include hill-climbing, simulated annealing, tabu search, ant colony methods, swarm intelligence and evolutionary methods [46].)
- Describe search-based testing approaches, which are inherently structural (white-box), functional (black-box) or grey-box (combination of structural and functional). Grey-box testing includes assertion testing and exception condition testing [182]. This exclusion criterion is relaxed to include those studies where a structural test criterion is used to test non-functional properties, e.g. [24].
- Are not related to the testing of the end product, e.g. [264].
- Are related to test planning, e.g. [64].
- Make use of model checking and formal methods, e.g. [68, 15].
- Report performance of a particular metaheuristic instead of its application to software testing, e.g. [154].
- Report on test case prioritization, e.g. [251].
- Are used for prediction and estimation of software properties, e.g. [30].

The first phase of research resulted in a total of 501 papers. After eliminating duplicates found by more than one electronic database, we were left with 404 papers. Table 5.1 shows the distribution of papers before duplicate removal among different sources.

Table 5.1: Distribution of papers before and after duplicate removal among different publication sources.

| Source | Count |
|---|---|
| IEEE Xplore | 209 (179) |
| EI Compendex | 140 (87) |
| ISI Web of Science | 61 (48) |
| ACM Digital Library | 58 (57) |
| Conferences and Journals | 33 (33) |
| Total | 501 (404) |

The exclusion was done using a tollgate approach (Figure 5.2). To begin with, a single researcher excluded 37 references out of a total of 404 primarily based on title and abstract, which were clearly out of scope and did not relate to the research

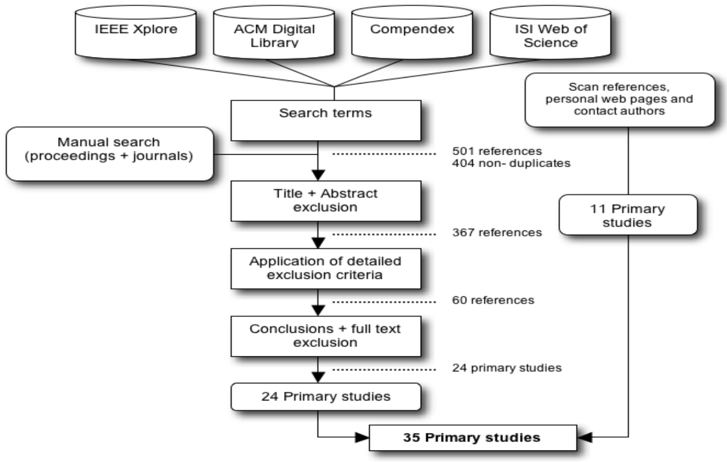

IEEE Xplore
ACM Digital Library
Compendex
ISI Web of Science
Search terms
Manual search (proceedings + journals)
Scan references, personal web pages and contact authors
501 references 404 non- duplicates
Title + Abstract exclusion
367 references
11 Primary studies
Application of detailed exclusion criteria
60 references
Conclusions + full text exclusion
24 primary studies
24 Primary studies
35 Primary studies

Figure 5.2: Multi-step filtering of studies (tollgate approach) and final number of primary studies.

question. The remaining 367 references were subject to detailed exclusion criteria, which involved three researchers. First, each researcher applied the exclusion criteria independently. Out of 367 references, the three researchers were in agreement on 229 references to exclude, 25 to include and 113 required a meeting to reach consensus. In the meeting, the researcher in the minority for a paper tried to convince others; otherwise the majority decision was taken. This application of detailed exclusion criteria resulted in 60 remaining references, which were further filtered out by reading the full-text. A final figure of 24 primary studies was reached after excluding similar studies that were published in different venues. The 24 primary studies were complemented with 11 more papers from phase 2 of the search strategy (Figure 5.1). The fact that we gathered 11 papers from phase 2 of the search strategy indicates that making a generic search string that would give the entire relevant set of primary studies from searching only within electronic databases is difficult in the field of study under investigation. The terminologies used by various authors differed a lot; both in terms of specifying the non-functional property and the used metaheuristic. As an example, if we consider the primary study [133], identified using phase 2, we observed that although it does mention using *genetic programming* in the title, it does not mention the target non-functional property of *security*. Similarly, in the abstract and key words, we do not find words synonymous to *testing*. Therefore, we believe that the phase 2 of the search strategy helped us to gather a more representative set of primary studies.

### 5.2.4 Study quality assessment and data extraction

Since we did not impose any restriction in terms of any specific research method or experimental design, therefore the study quality assessment covered both quantitative and qualitative studies. The quality data can be used to devise a detailed inclusion/exclusion criteria and/or to assist data analysis and synthesis [150]. We applied the study quality assessment primarily as a means to guide the interpretation of findings for data analysis and synthesis [150], so as to avoid any misinterpretation of results due to study quality. We did not assign any scores to the criterion (because our aim was not to rank studies according to an overall quality score) but used a binary 'yes' or 'no' scale [76]. Table B.1 in Appendix B shows the application of the study quality assessment criteria where a ($\surd$) indicates 'yes' and ($\times$) indicates 'no'. Most of our developed criteria were fulfilled by all of the studies, exceptions being [94, 91, 36, 24, 206] where evidence of a comparison group was missing, but these quality differences were not found to be largely confounded with study outcomes. What follows next is the list of developed criteria:

- Is the reader able to understand the aims of the research?
- Is the context of study clearly stated, that includes population being studied (e.g. academic vs. industrial) and tasks to be performed by population (e.g. small scale vs. large scale)
- Was there a comparison or control group?
- Are the measures used in the study fully defined [150]?
- Is there an adequate description of the data collection methods?
- Does the data collection methods relate to the aims of the research?
- Is there a description of the method used to analyze data?
- Are the findings clearly stated and relate to the aims of research?
- Is the research useful for software industry and research community?
- Do the conclusions relate to the aim and purpose of research defined?

We designed a data extraction form to collect information needed to address the review questions and data synthesis. Study quality data was not part of data extraction form as it was assessed separately. To assess the consistency of data extraction, a small sample of primary studies were used to extract data for the second time. In addition to the standard information of title, author(s), journal and publication details; the data extraction form included information about main theme of study, motivation for the main theme, type of non-functional testing addressed, type of metaheuristic search technique used, examples of application of approach, constraints/limitations in the application of the metaheuristic search technique, identified areas of future research and major conclusion. For each primary study, we further extracted the information relating to the method of evaluation, number of test objects, performance factor evaluated and the experimental outcomes.

## 5.3 Results and synthesis of findings

In this section we describe the descriptive evaluation of the assessed literature in relation to the research questions. The 35 primary studies were related to the application of metaheuristic search techniques for testing five non-functional properties: execution time, quality of service (QoS), security, usability, and safety. The number of primary studies describing each non-functional property is: 15 (execution time), 2 (quality of service), 7 (security), 7 (usability) and 4 (safety). Relevant information describing the distribution of primary studies within each non-functional property is shown in Table 5.2.

Figure 5.3 shows the year-wise distribution of primary studies within each non-functional property as well as the frequency of application of different metaheuristics [9]. The bubble at the intersection of axes contains the actual number of contribution(s). It is evident from the figure that genetic algorithms are the most widely used metaheuristic with applications in 21 papers across different types of non-functional testing. In the left portion of Figure 5.3, each bubble represents the actual number of primary studies within each non-functional area in respective years from 1996–2007. More details on Figure 5.3 can be found in [9] which is a systematic mapping study, giving a broad overview of studies without reviewing the studies in detail.

### 5.3.1 Execution time

The application of evolutionary algorithms to test real-time requirements in embedded computer systems involves finding the best and worst case execution times (BCET, WCET) to determine if timing constraints are fulfilled. A violation of the timing con-

Table 5.2: Distribution of primary studies per non-functional area.

| Non-functional property | Author(s) | Year | Reference |
|---|---|---|---|
| Execution time (42.86%) | Wegener et al. | 1996 | [252] |
| | Alander et al. | 1997 | [14] |
| | Wegener et al. | 1997 | [255] |
| | Wegener et al. | 1998 | [253] |
| | O'Sullivan et al. | 1998 | [203] |
| | Tracey et al. | 1998 | [245] |
| | Mueller et al. | 1998 | [185] |
| | Puschner et al. | 1998 | [211] |
| | Pohlheim et al. | 1999 | [207] |
| | Wegener et al. | 2000 | [254] |
| | Groß et al. | 2000 | [94] |
| | Groß | 2001 | [91] |
| | Groß | 2003 | [92] |
| | Briand et al. | 2005 | [36] |
| | Tlili et al. | 2006 | [243] |
| Quality of service (5.71%) | Canfora et al. | 2005 | [51] |
| | Di Penta et al. | 2007 | [69] |
| Security (20%) | Dozier et al. | 2004 | [75] |
| | Kayacik et al. | 2005 | [132] |
| | Budynek et al. | 2005 | [44] |
| | Del Grosso et al. | 2005 | [96] |
| | Kayacik et al. | 2006 | [131] |
| | Kayacik et al. | 2007 | [133] |
| | Del Grosso et al. | 2007 | [95] |
| Usability (20%) | Stardom | 2001 | [233] |
| | Cohen et al. | 2003 | [55] |
| | Cohen et al. | 2003 | [56] |
| | Cohen et al. | 2003 | [174] |
| | Nurmela | 2004 | [196] |
| | Shiba et al. | 2004 | [225] |
| | Bryce et al. | 2007 | [42] |
| Safety (11.43%) | Tracey et al. | 1999 | [246] |
| | Abdellatif-Kaddour et al. | 2003 | [2] |
| | Baresel et al. | 2003 | [24] |
| | Pohlheim et al. | 2005 | [206] |

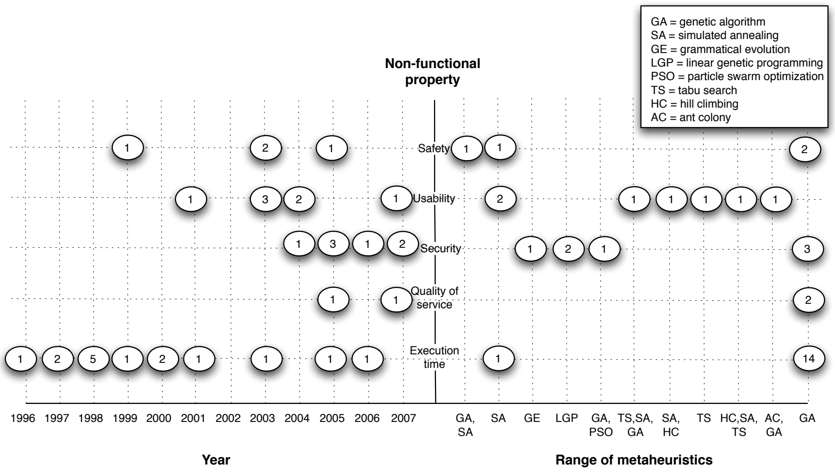


Figure 5.3: Distribution of NFSBST research over range of applied metaheuristics and time period (adapted from [9]).

straint or temporal error means that either the outputs are produced too early, or their computation takes too long [253]. The use of evolutionary computation to find the input situations causing longest or shortest execution times is an example of evolutionary testing. Evolutionary testing is seen as a promising approach for verifying timing constraints and a number of studies proving the efficacy of the approach can be found in literature. This dynamic approach to verify timing constraints involves testing the run-time behavior of an embedded system based on execution in an application environment. Testing of real-time systems is found to be costly as compared to conventional applications as additional requirements of timeliness, simultaneity and predictability needs to be tested. Although there are numerous methods to test logical correctness, the lack of support for testing temporal behavior [185] motivated the use of evolutionary computation in testing extreme execution times.

In [252], genetic algorithms (GA) were used to search for input situations that produce very long or very short execution times. The fitness function used was the execution time of an individual measured in micro seconds. The experimental results using a simple C function showed that the longest execution time of 26.27 $\mu$sec was found very quickly with GA in less than 20 generations. Moreover, a new shortest execution time of 5.27 $\mu$sec, which was not discovered by statistical and systematic testing, was found. This study marks one of the earliest use of GA to test temporal correctness of real-time systems.

Alander et al. [14] presented experiments performed in a simulator environment to measure response time extremes of protection relay software using genetic algorithms. The fitness function used was the response time of the tested software. The results showed that GA generated more input cases with longer response times. In [255], experiments were performed using genetic algorithms involving five test objects from different application domains having varying lines of code and integer input parameters.

This time the fitness function used was the execution time measured in terms of processor cycles rather than seconds. The results show that GA consistently outperformed random testing by finding more extreme times.

The research community soon realized the benefits of measurement in terms of processor cycles; being more precise and independent of the interrupts from the operating system (e.g. context switching and paging). Also measurement in terms of processor cycles is deterministic in the sense that it is independent of system load and results in the same execution times for the same set of input parameters. However, such a measurement is dependent on the compiler and optimizer used, therefore, the processor cycles differ for each platform [255].

Wegener and Grochtmann [253] continued further with experimentation to compare evolutionary testing (using genetic algorithms) with random testing. The fitness function used was duration of execution measured in processor cycles. This time the range of input parameters was raised to 5,000. The results showed that, with a large number of input parameters, evolutionary testing obtained more extreme execution times with less or equal testing effort than random testing. In order to better evaluate the application of evolutionary testing, Groß et al. [93] presented the design of an operational experimental environment for evolutionary testing with the integration of a commercially available timing package.

The aforementioned experimental results identified several limitations when using evolutionary testing. The instrumentation required for the software under test (SUT), which extends the executable programming code by inserting hardware dependent counting instructions, is bound to affect the execution times. Also as a typical search strategy, it is difficult to ensure that the execution times generated in the experiments represents global optimum. More experimentation is also required to determine the most appropriate and robust parameters. Lastly, there is a need for an adequate termination criterion to stop the search process.

In [203], cluster analysis was used on the population from the most recent generation to determine if GA should terminate. Execution time measured in processor cycles was used as a fitness function and a complex algorithm from the domain of automotive electronics was used for seven runs of GA. Cluster analysis was performed on the final population (generation 399) of each run. With cluster analysis, it was possible to examine which of the test runs converged to local optima and thus continuing with these runs would not yield better results. The results of the study demonstrated the potential of incorporating cluster analysis as a useful termination criterion for evolutionary tests and suggested appropriate changes in the search strategy to include cluster analysis information.

Tracey et al. [245] applied simulated annealing (SA) to test four simple programs for WCET as part of a generalized test data generation framework. The fitness function used is the measure of actual dynamic execution time. The WCET of the programs was already known and a valid test case was one that exercised a path yielding the already known WCET. The results of the experiment showed that the use of SA was more effective with larger parameter space. The authors highlighted the need of a detailed comparison of various optimization techniques to explore WCET and BCET of the SUT. With this goal in mind, Pohlheim and Wegener [207] used an extension of genetic algorithms making use of multiple sub-populations, each using a different search strategy. The authors name the approach as extended evolutionary algorithms. The duration of execution measured in processor cycles was taken as the fitness function. The extended evolutionary algorithm was applied on two test objects. The first test object was the bubble sort algorithm and the results from this experiment was used to find appropriate evolutionary parameters for the second test object which contained software modules from a motor control project. The evolutionary algorithm found longer execution times for all the given modules in comparison with systematic testing.

As mentioned earlier, it is difficult to ensure that the execution times generated in the experiments represent a global optimum. Therefore it appears interesting to compare the results of evolutionary testing with static analysis to find a bound within which WCET and BCET might lie. Mueller et al. [185] presented such a comparison. Both approaches were used in five experiments to determine the WCET and BCET of different programs. Three programs were from real-time systems while the remaining two were general-purpose algorithms. The fitness function used is the execution time measured in processor cycles. The results showed that methods of static analysis and evolutionary testing bound the actual execution times. For WCET, the estimates of static analysis provided an upper bound while the measurements of evolutionary testing yielded a lower bound. Conversely, static analysis' estimates provided a lower bound for BCET while evolutionary testing measurements constituted an upper bound. In [211], genetic algorithms were applied to find WCET for seven programs and the results were compared with those from random search, static analysis and best effort timings that were researchers' own efforts to find input data to yield WCET. The execution time measured in processor cycles as well as time units were used as a fitness function for different programs. Genetic algorithms found the same or longer times than random search. In comparison with best effort timings, genetic algorithms matched the timings and found a longer time in one case, while in comparison with static analysis, the upper bounds were not broken but were matched on several occasions. In another study, Wegener et al. [254] used genetic algorithms to test temporal behavior of six time critical tasks in an engine control system, with the fitness function used was execution time measured in processor cycles. Genetic algorithms outperformed both random search and developer-made tests.

In [92], 15 example test programs were used in experiments to measure the maximal execution times using genetic algorithms. The fitness function used was the execution time of the test object for a particular input situation measured in microseconds. The results of evolutionary testing were compared with random testing and with the performance of an experienced human tester. The results indicated that evolutionary testing outperforms random testing as random testing could only produce about 85% of the maximum execution times found by evolutionary testing. The human tester was more successful in 4 out of 15 test programs, which indicated the presence of properties of test objects that inhibit evolutionary testability [91] i.e. the ability of an evolutionary algorithm to successfully generate test cases that violates the timing specification.

Groß et al. [94] presented a prediction model based on complexity of the test object, which can be used to predict evolutionary testability. It was found that there were several properties inhibiting evolutionary testability, which included small path domains, high-data dependence, large input vectors, and nesting.

Additionally, several source code measures, which map program attributes inhibiting evolutionary testability, were also presented in [91]. Code annotations were inserted into the test object's source code along their shortest and longest algorithmic execution paths. The fitness function was then based on maximal and minimal possible annotation coverage by the generated input situations. The individual measures were combined to form a prediction system that successfully forecasted evolutionary testability with 90% accuracy.

Results from the two studies [94, 91] also confirmed that there is a relationship between the complexity of a test object and the ability of evolutionary algorithm to produce input parameters according to B/WCET. The results also confirmed the properties (given above) of the test programs that caused most problems for evolutionary testing. Due to program properties inhibiting evolutionary testability, [92] pointed out that an ideal testing strategy is a combination of evolutionary testing supported by human knowledge of the test object. The initial population of individuals can benefit from human knowledge to direct the search in those areas of search space that are difficult to reach as the fitness function does not provide information to generate such unlikely input combinations.

In one of the more recent studies, Tlili et al. [243] used the approach of seeding an evolutionary algorithm with test data achieving a high structural coverage and reduction in the amount of search space by restricting the range of input variables in the initial population. The fitness function was the measurement of the execution time of test data as number of CPU clock ticks. The results indicated that for almost all the test objects, application of seeding and range restriction outperform standard evolutionary real-time testing with random initial population when measuring long execution times. Also with seeding and range restriction, fewer generations found the longest execution times. Similar results were achieved for finding the shortest execution times.

In [36, 37], another approach to use genetic algorithms for critical deadlines misses was used. The authors called the approach stress testing because the system was exercised in such a way that some tasks were close to missing a deadline. This approach can also be called robustness testing but since the basic objective of the paper was to find the sequence of arrival times of events for aperiodic tasks, which will cause the greatest delays in the execution of the target task, we chose to discuss this paper under execution time. The study was restricted to seeding times for aperiodic tasks and the tasks synchronization, since input data were accounted for in execution time estimates. In comparison with other approaches to evolutionary testing for finding the WCET, this approach was different in the sense that test data design did not require the implementation of the system and, secondly, did not consider the tasks in isolation. Genetic algorithms were used to search for the sequence of arrival times of events for aperiodic tasks that would cause the greatest delays in execution of the target task. The fitness function was expressed in an exponential form, based on the difference between the deadline of an execution and the executions actual completion. A prototype tool called Real Time Test Tool (RTTT) was built to facilitate the execution of runs of genetic algorithm. Two case studies were conducted; one case study consisted of researchers own scenarios while the second consisted of an actual real-time system. The results from the timing diagrams illustrated that RTTT was a useful tool to stress the system more than the scenarios covered by schedulability theory.

A summary of results of applying metaheuristics for testing temporal properties is given in Table 5.3.

### 5.3.2 Quality of Service

Search-based testing of QoS (Quality of Service) represents a mix of search-based software engineering and service-oriented software engineering. Metaheuristic search techniques have been used for quality of service aware composition and violation of service level agreements (SLAs) between the integrator and the end user.

In [51], genetic algorithms were used to determine the set of service concretizations (i.e. bindings between abstract and concrete services) that lead to QoS constraint

Table 5.3: Summary of results applying metaheuristics for testing temporal properties. The last column on the right covers any issues such as constraints, limitations and highlights.(GA is short for Genetic Algorithm, SA is short for Simulated Annealing while EGA is short for Extended GA.)

| Article | Applied meta-heuristic | Fitness function used | Limitations and highlights |
|---|---|---|---|
| Wegener et al. 1996 [252] | GA | Exec. time, microseconds | **Instrumentation** of the test objects causes probe effects. **The** execution times do not always represent global optimum. |
| Alander et al. 1997 [14] | GA | Exec. time, milliseconds | **The** experiments are performed in a simulator environment.**Non-determinism** of the fitness function is problematic. |
| Wegener et al. 1997 [255] | GA | Exec. time, processor cycles | **The** decision about when to stop the search is arbitrary. |
| Wegener and Grochtmann 1998 [253] | GA | Exec. time, processor cycles | **There** is a need to find most appropriate and robust parameters for evolutionary testing. **Cluster** analysis can be a used as a measure for termination of search. |
| O'Sullivan et al. 1998 [203] | GA | Exec. time, processor cycles | **The** search strategy needs to make use of cluster analysis to react to stagnations. |
| Tracey et al. 1998 [245] | SA | Exec. time, time units | **Ways** to reduce the amount of search-space is useful. **There** is a need to devise different software metrics to guide the search. |
| Pohlheim and Wegener 1999 [207] | EGA | Exec. time, processor cycles | **A** combination of systematic and evolutionary testing is required for thoroughly testing real-time systems. **Instead** of random generation of initial population, use of testers knowledge improves search performance. |
| Mueller and Wegener 1998 [185] | GA | Exec. time, processor cycles | **For** WCET, estimates of static analysis provide an upper bound, evolutionary testing gives a lower bound. **For** BCET, static analysis estimates provide a lower bound, evolutionary testing constitutes an upper bound. |
| Puschner and Nossal 1998 [211] | GA | Exec. time, processor cycles & time units | **Further** investigation is required to escape the large plateaus of equal fitness function values. |
| Wegener et al. 2000 [254] | GA | Exec. time, processor cycles | **Static** analysis techniques can support evolutionary testing in search space reduction. |
| Groß 2003 [92] | GA | Exec. time, microseconds | **Evolutionary** testability is inhibited by source code properties of small path domains and high-data dependence. |
| Groß 2001 [91] | GA | Coverage of code annotations along shortest and longest execution paths | **The** measures did not cater for the reliance on the parameter setting of the GA. **Effects** of underlying hardware are not taken into account. **The** number of considered data samples is low. |
| Groß et al. 2000 [94] | GA | Coverage of code annotations along shortest and longest execution paths | **Traditional** design principles of low coupling and high cohesion are an important issue for an evolutionary approach. |
| Tlili et al. 2006 [243] | EGA | Exec. time, processor cycles | **The** nature of the test objects is not evident in the study. **Using** branch coverage as a criterion for seeding has the limitation that it does not handle the execution of all the possible values of the predicates forming the conditions. |
| Briand et al. 2005 [36] | GA | Exponential fitness function based on the difference between executions deadline and executions actual completion | **The** termination criterion is not adaptive and is taken as fixed number of generations. **The** specification of test cases does not require any running implementation of system. **It** takes into account tasks synchronizations. |

satisfaction while optimizing the objective function. An abstract service is the feature required in a service orchestration while concrete services represent functionally equivalent services realizing the required feature. In a contract with potential users, the service provider can estimate ranges for the QoS attributes as part of Service Level Agreement (SLA), i.e. a contract between an integrator and end-user for a given QoS level. QoS attributes consist of non-functional properties such as cost, response time and availability so the fitness function optimized the QoS attribute chosen by the service integrator. The QoS attributes of composite services were determined using rules with aggregation function for each workflow construct. The fitness function was designed in a way to maximize some QoS attributes (e.g. reliability and availability) while minimizing others (e.g. cost and response time). Based on the type of penalty factor (static vs. dynamic), static fitness function and dynamic fitness functions were proposed. With experiments on 18 invocations of 8 distinct abstract services, the performance of genetic algorithms was compared with integer programming. The results showed that when the number of concrete services is small, integer programming outperformed genetic algorithms. But as the number of concrete services increased, the genetic algorithm was able to keep its time performance while integer programming grew exponentially.

Di Penta et al. [69] used genetic algorithms to generate test data that violated QoS constraints causing SLA violations. The generated test data included combinations of inputs and bindings for the service-oriented system. The test data generation process was composed of two steps. In the first step, the risky paths for a particular QoS attribute were identified and in the second step, genetic algorithms were used to generate test cases that covered the path and violated the SLA. The fitness function combined a distance-based fitness that rewards solutions close to QoS constraint violation, with a fitness guiding the coverage of target statements. The two fitness factors were dynamically weighted. The approach was applied to two case studies. The first case study was an audio processing workflow containing invocations to four abstract services. The second case study, a service producing charts, applied the black-box approach with fitness calculated only on the basis of how close solutions violate QoS constraint. In case of audio workflow, the genetic algorithm using the proposed fitness function, which combined distance-based fitness with coverage of target statements, outperformed random search. For the service producing charts, use of black box approach successfully violated the response time constraint, showing the violation of QoS constraints for a real service available on the Internet.

A summary of results of applying metaheuristics for QoS-aware composition and violation of SLA is given in Table 5.4.

Table 5.4: Summary of results applying metaheuristics for QoS-aware composition and violation of SLA. The last column on the right covers any issues such as constraints, limitations and highlights.(GA is short for Genetic Algorithm.)

| Article | Applied meta-heuristic | Fitness function used | Limitations and highlights |
|---|---|---|---|
| Canfora et al. 2005 [51] | GA | Based on the maximization of desired QoS attributes while minimizing others, including a static or dynamic penalty function | **The** QoS attributes of component services needs to be computed for workflow constructs. **The** fitness function needs to incorporate the constraints of balancing different QoS attributes. Also weights need to be assigned to a particular QoS attribute to indicate the importance. |
| Di Penta et al. 2007 [69] | GA | Combination of distance based fitness that rewards solutions close to QoS constraint violation with a fitness guiding the coverage of target statements | **The** study does not deal with the dependency of violation of some QoS attributes on the network and server load. **Instrumentation** of the workflow can cause probe effects which can cause the deviation of fitness calculation. |

## 5.3.3 Security

A variety of metaheuristic search techniques have been applied to detect security vulnerabilities like detecting buffer overflows; including grammatical evolution, linear genetic programming, genetic algorithm and particle swarm optimization.

In [132], grammatical evolution (GE) was used to discover the characteristics of a successful buffer overflow. The example vulnerable application in this case performed a data copy without checking the internal buffer size.

The exploit was represented by a sample C program that approximated the desired return address and assembled the malicious buffer exploit. The malicious buffer contained a shell code, representing attacker's arbitrary code that overwrites the return address to gain control.

For the attack to be successful, it was important to jump to the first instruction of the shell code or to the sequence of no operation instructions called NoOP sled. The fitness function represented six characteristics of malicious buffer which included existence of the shell-code, success of the attack, NoOP sled score, back-to-back desired return addresses, desired return address accuracy and score calculated on NoOP sled size. Three sets of experiments were performed, namely basic Grammatical Evolution (GE), GE with niching (to include population diversity) and GE with niching and NoOP minimization. The results found were comparable. In [131], the vulnerable system call was taken to be the UNIX `execve` command and linear GP was used to evolve variants of an attack. The UNIX `execve` command required the registers `EAX`, `EBX`, `ECX`, `EDX` to be correctly configured and the stack to contain the program name to be executed. The fitness function returned a maximum fitness of 10 if all conditions were satisfied.

The experimental results indicated that evolved attacks discovered different ways of attaining sub-goals associated with building buffer overflow attacks and expanding the instruction set provided better results as compared to basic GP.

Kayacik et al. [133] used linear GP for automatic generation of mimicry attacks to perform evasion of intrusion detection system (IDS), which in this case was an open source target anomaly detector called `stride`. The candidate mimicry attacks were in the form of system call sequences. The system call sequences consisted of most frequently executed instructions from the vulnerable application, which in this case is `traceroute`, a tool used to determine the route taken by packets across an IP network.

An acceptable anomaly rate was established for `stride`. The objective of the attacker therefore was to reduce the anomaly rate below this acceptable limit. The study described an attack denoted by successful completion of three steps i.e. open a UNIX password file, write a line and close the file. The fitness function rewarded attacks that successfully followed the steps and at the same time minimized the anomaly rate. The results showed that the approach was able to reduce the anomaly rate to $\sim$2.97% for the entire attack.

In [75], security vulnerabilities in an artificial immune system (AIS) based IDS were identified using genetic algorithms and particle swarm optimization. The study used GENERITA Red Team (GRT), a system based on evolutionary algorithms, which performed the vulnerability analysis of the IDS. The AIS-based IDS communicates with the GRT by receiving red packets in the form of attacks and returns the percentage of the detector set that failed to detect the red packet. This percentage was the fitness of the red packet. The packets took the form of triplets (`ip_address`, `port`, `src`) while the AIS maintained a population of detectors with different ranges of IP addresses and ports. Matching rules were applied to match the data triple and a detector. The GRTs used consisted of a steady-state GA and six variants of particle swarm optimization. Experiments were performed using data representing 35 days of simulated network traffic. The results showed that genetic algorithms outperform all of the swarms with respect to the number of distinct vulnerabilities discovered.

Budynek et al. [44] modeled the hacker behavior along with the creation of a hacker grammar for exploring hacker scripts using genetic algorithms. A hacker script contained sequences of UNIX commands issued by the hacker upon logging in to the system. One script was one individual with a single UNIX command acting as a gene. A fitness function was defined based on the efficiency and effectiveness of the hacking scripts i.e. the script fitness value was calculated by number of goals achieved, number of pieces of evidence discovered by the log analyzer, number of bad commands used by the hacker and the length of the script used by the hacker. The results of experiments showed various top-scoring scripts obtained from various runs.

Grosso et al. [96] used static analysis and program slicing to identify vulnerable statements and their relationships, that were further explored using genetic algorithms for buffer overflows. Three different fitness functions were compared. The first one (vulnerable coverage fitness) included weighted values for statement coverage, vulnerable statement coverage and number of executions of vulnerable statements. The second fitness function (nesting fitness) incorporated observed maximum nesting level corresponding to the current test case while the third fitness function (buffer boundary fitness) included a term accounting for the distance from the buffer boundaries. Two programs were used for experimentation. The case in which the expert's knowledge was used to define the initial search space and also for the case having random initial population showed that buffer boundary fitness outperformed both the vulnerable coverage and the nesting fitness. This showed that fitness functions using distance from the limit of buffers were helpful for deriving genetic algorithm evolution. In [95], the authors improved on the previous basic boundary fitness [96] to propose a dynamic weight fitness in which the genetic algorithm weights were calculated by solving a maximization problem via linear programming. So with weights that could be tuned at each genetic algorithm generation, fast discovery of buffer overflows could be achieved. The dynamic weight fitness outperformed the previous basic boundary fitness on experiments with two different sets of C applications.

A summary of results of applying metaheuristics for detecting security vulnerabilities is given in Table 5.5.

### 5.3.4 Usability

Usability testing in the context of application of metaheuristics is concerned with construction of covering array, which is a combinatorial object.

The user is involved in numerous interactions taking place through the user interface. With the number of different features available and their respective levels, the interactions cannot be tested exhaustively due to a combinatorial explosion. Interaction testing offers savings, in that it aims to cover every combination of pair-wise (or $t$-way) interaction at least once [41]. It is interesting to see that there are different competing constraints. On one hand, the objective is high-coverage, while on the other hand the test suite size needs to be small to reduce overall testing cost. Covering array (CA) needs to be constructed to capture the $t$-way interactions. For software systems, each factor (feature or component) comprises of different levels (options or parameters or values), therefore a mixed level covering array (MCA) is proposed. However, as compared to CA, there are few results on the upper bound and construction algorithms for mixed level covering arrays, especially using heuristic search techniques [55]. Algebraic constructs, greedy methods and metaheuristic search techniques have been applied to construct covering arrays.

Table 5.5: Summary of results applying metaheuristics for detecting security vulnerabilities. The last column on the right covers any issues such as constraints, limitations and highlights. (GE is short for Grammatical Evolution, LGP is short for Linear Genetic Programming and PSO is short for Particle Swarm Optimization.)

| Article | Applied meta-heuristic | Fitness function used | Limitations and highlights |
|---|---|---|---|
| Kayacik et al. 2005 [132] | GE | Representation of six characteristics of malicious buffer reflecting multiple behavioral objectives | **The** shell code or the attackers arbitrary code needs to be modified to increase the success chances of malicious buffer. |
| Kayacik et al. 2006 [131] | LGP | Fitness function based on the configuration of registers and stack | **A** mechanism for maintaining diversity of population is necessary. |
| Kayacik et al. 2007 [133] | LGP | Evaluation in terms of completion of steps leading to an attack and minimization of anomaly rate | **The** proposed approach is dependent on the core attack which is then used to create mimicry attacks. **The** approach is also dependent on the set of permitted system calls as defined by the user. |
| Dozier et al. 2004 [75] | GA, PSO | Percentage of the detector set that failed to detect the red packet from GRT | **An** attack is not as such constructed but is represented as a triple packet. |
| Budynek et al. 2005 [44] | GA | The fitness is calculated based upon the scripts ability of how much damage it can inflict with the most compact possible sequence of commands | **The** goals of the hacker script grammar needs to be defined beforehand. |
| Grosso et al. 2005 [96] | GA | Three different fitness functions covering vulnerable statements, maximum nesting level and buffer boundary | **Dependency** on tools for static analysis and program slicing. **Use** of instrumentation is a probable obstacle in expanding the approach for larger case studies. |
| Grosso et al. 2007 [95] | GA | A dynamic weight fitness function in which the weight determination is a maximization problem | **Dependency** on tools for static analysis and program slicing. **Use** of instrumentation is a probable obstacle in expanding the approach for larger case studies. **Additional** computational time required for linear programming calculation. |

Our interest here is to explore the use of metaheuristic search techniques for constructing covering arrays. Hoskins et al. provide definitions relevant to covering array and mixed level covering array [110]:

> *A covering array, $CA\lambda(N;t,k,v)$, is an $N \times k$ array for which every $N \times t$ sub-array has the property that every $t$-tuple appears at least $\lambda$ times. In this application, $t$ is the strength, $k$ is the number of factors (degree), and $v$ is the number of symbols for each factor (order). When $\lambda$ is 1, every $t$-way interaction is covered at least once; this is the case of most interest, and we often omit the subscript $\lambda$ when it is 1. The covering array is optimal if it contains the minimum possible number of rows. The size of such a covering array is the covering array number $CAN(t,k,v)$.*
>
> *A mixed level covering array, $MCA\lambda(N;t,k,(v_1,v_2,\ldots,v_k))$, is an $N \times k$ array in which, for each column i, there are exactly $v_i$ levels; again every $N \times t$ sub-array has the property that each possible $t$-tuple occurs at least $\lambda$ times. Again $\lambda$ is omitted from the notation when it is 1. A mixed covering array provides the flexibility to construct test suites for systems in which components are not restricted to having the exact same number of levels.*

To adapt to the practical concerns of software testing, it is desirable that some subset of features have higher interaction coverage. For example, the overall system might have 100% two-way coverage, but a subset of features might have 100% three-way coverage. To this end, in [55], Cohen et al. propose variable strength covering arrays. As with mixed level covering arrays, construction methods and algorithms for variable strength test suites is in its preliminary stages with [55] providing some initial results for test suite sizes constructed using simulated annealing (SA).

With respect to the application of metaheuristics, the fitness function used for constructing a covering array is the number of uncovered $t$-subsets, so the covering array itself will have a cost of 0. Since one does not know the size of the test suite *a priori*, therefore, heuristic search techniques apply transformations to a fixed size array until constraints are satisfied. The results of implementing SA for handling $t$-way coverage in fixed level cases are provided by [55]. The results showed that in comparison with greedy search techniques used in Test Case Generator (TCG) [248] and Automatic Efficient Test Generator (AETG) [54], SA improved on the bounds of minimum test cases in a test suite of strength two, e.g. for $MCA(N;2,5^1 3^8 2^2)$, SA gave 15 as minimum test cases as compared to 20 and 19 by TCG and AETG respectively. In case of strength three constructions, the SA algorithm did not perform as well as the algebraic constructions. Therefore, the initial results indicated SA as more effective than other

Table 5.6: Variants of approaches used for constructing covering arrays using metaheuristics.

| Approach used | Articles |
|---|---|
| Independent application of metaheuristics | Cohen et al. [55, 56], Stardom [233], Nurmela [196], Shiba et al. [225] |
| Use of an integrated approach | Cohen et al. [174], Bryce et al. [42] |

approaches for finding smaller sized test suites. On the other hand, SA took much more execution time as compared to simpler heuristics. Stardom [233] also used SA, GA and tabu search (TS) for constructing covering arrays. The results indicated that SA and TS were best in constructing covering arrays. A genetic algorithm turned out to be least effective; taking more time and moves to find good covering arrays. Stardom reported new upper bounds on size of covering array using SA, some of which were later improved by [55]. Stardom's study indicated that SA's main advantage was the capability of executing many moves in a short time; therefore if the search space was dense, SA quickly located objects. On the other hand, TS performed much better when the size of an array's neighborhood was smaller.

Along with the application of metaheuristic techniques for constructing covering arrays, there is also evidence of integrated approaches (Table 5.6). Cohen et al. [55] proposed one such approach for using algebraic construction along with search techniques. An example of this integrated approach is given in [174] where a new strategy called augmented annealing takes advantage of computational efficiency of algebraic construction and generality of SA. Specifically, algebraic construction reduced a problem to smaller sub-problems on which SA runs faster. The experimental results reported new bounds for some strength three covering arrays e.g. $CA(3,6,6)$ and $CA(3,10,10)$. A hybrid approach is also given in [42] for constructing covering array. The study focussed on covering as many $t$-tuples as early as possible. So rather than minimizing the number of tests to achieve $t$-way coverage, the initial rate of coverage was the primary concern. The hybrid approach applied a one-test-at-a-time greedy algorithm to initialize tests and then applied heuristic search to increase the number of $t$-tuples in a test. The heuristic search techniques applied were hill-climbing, SA, TS and great flood. With different inputs, SA in general produced the quickest rate of coverage with 10 or 100 search iterations. The study also concluded that smaller test suites do not relate to greater rate of coverage; hence two different and sometimes inconsistent goals when applied in a real world setting.

As mentioned earlier, there is less evidence on construction of variable strength arrays. One such study used SA to find variable strength arrays and provided initial bounds [56]. In his work using TS, Nurmela improved on previously known upper bounds on the sizes of optimal covering arrays [196]. Experimenting with number of factors and number of values for each factor, good upper bounds were tabulated for strength-two covering arrays. In addition, the study improved upper bounds for strength three covering arrays. The TS algorithm was found to work best for strength two covering array with number of levels for each factor equal to three. According to [196], it was difficult to be sure about the upper bounds to be optimal or not because TS being a stochastic algorithm could improve upon the new bounds if given more computing time.

Toshiaki et al. used genetic algorithms (GA) and ant colony algorithm (ACA) for constructing covering arrays [225]. The results were compared with AETG, In-Parameter Order (IPO) [162] algorithm and SA. SA outperformed their results of using GA and ACA with respect to the size of resulting test sets for two-way and three-way testing. Their results however outperformed AETG for two-way and three-way testing. It was interesting to find that using GA, the results did not match with those produced by Stardom's study [233] which indicated that GA did not perform well in generating covering arrays, even though several attempts were made to modify the structure of the algorithm.

A summary of results of applying metaheuristics for covering array construction is given in Table 5.7.

### 5.3.5 Safety

Safety testing is an important component of the testing strategy of safety critical systems where the systems are required to meet safety constraints. In terms of metaheuristic search techniques, SA and GA are applied for safety testing.

In [246], the authors proposed an approach using GAs and SA to generate test data violating a safety property. This approach extended the authors' previous work in developing a general framework for dynamically generating test data. The violation of a safety property meant a hazard or a failure condition, which was initially identified using some form of hazard analysis technique e.g. functional hazard analysis. The fitness function used evaluates different branch predicates and evaluates to zero if the safety property evaluates to false and will be positive otherwise. The search stopped when a test data with a zero cost was found. The cost function calculation is presented in Table 5.8. where $K$ represents the penalty which was added for undesirable data [246]. The paper provided a simple example where either SA or GAs can be applied to automatically search for test data violating safety properties that must hold 'after' the execution of the SUT. The same given cost function can also be used to generate test data to violate safety conditions at specific points 'during' the execution of the SUT.

Table 5.7: Summary of results applying metaheuristics for covering array construction. The last column on the right covers any issues such as constraints, limitations and highlights. (TS is short for Tabu Search, SA is short for Simulated Annealing, HC is short for Hill Climbing, ACA is short for Ant Colony Algorithm and GA is short for Genetic Algorithm.)

| Article | Applied meta-heuristic | Fitness function used | Limitations and highlights |
|---|---|---|---|
| Stardom 2001 [233] | TS, SA and GA | Number of uncovered $t$-subsets | **GA** takes more time and more moves to find a good covering array. **Larger** parameter sets require greater memory to store information. |
| Cohen et al. 2003 [55] | SA and HC | Number of uncovered $t$-subsets | **There** is still no best method for building variable strength test suite. **Combining** algebraic constructions with metaheuristic search is promising. |
| Cohen et al. 2003 [56] | SA | Number of uncovered $t$-subsets | **Variable** strength arrays guarantee a minimum strength of overall coverage and allow varying the strength among disjoint subsets of components. |
| Nurmela 2003 [196] | TS | Number of uncovered $t$-subsets | **If** more computing time is given, many of the new bounds can be improved slightly. |
| Cohen et al. 2003 [174] | SA | Number of uncovered $t$-subsets | **A** tool can be designed to take advantage of combining combinatorial construction along with heuristic search. |
| Bryce et al. 2007 [42] | TS, SA and HC | Number of uncovered $t$-subsets | **SA** provides the fastest rate of t-tuple coverage while TS is slowest. |
| Toshiaki et al. 2004 [225] | ACA, GA | Number of uncovered $t$-subsets | **The** test sets generated are small but they are not always optimal. |

Table 5.8: Cost function calculation.

| Element | Value |
|---|---|
| Boolean | `if TRUE then 0 else K` |
| $a = b$ | `if abs(a - b) = 0 then 0 else abs(a - b) + K` |
| $a \neq b$ | `if abs(a - b)` $\neq$ `0 then 0 else K` |
| $a < b$ | `if a - b < 0 then 0 else (a - b) + K` |
| $a \leq b$ | `if a - b` $\leq$ `0 then 0 else (a - b) + K` |
| $a > b$ | `if b - a < 0 then 0 else (b - a) + K` |
| $a \geq b$ | `if b - a` $\leq$ `0 then 0 else (b - a) + K` |
| $a \vee b$ | `min(cost(a), cost(b))` |
| $a \wedge b$ | `cost(a) + cost(b)` |
| $\neg a$ | Negation propagated over $a$ |

In this case, the SUT needed to be instrumented such that the branch predicates were replaced by procedures which served two purposes of returning the Boolean value of the predicate they replaced and adding to the overall cost the contribution made by each individual branch predicate that was executed. An example was given with an original program and the instrumented program with examples of how the fitness function was able to guide the search.

The approach presented by [246] has been extended by Abdellatif-Kaddour et al. [2] for sequential problems. In [2], SA was used for step-wise construction of test scenarios (progressive exploration of longer test scenarios) to test safety properties in cyclic real-time control systems. The stepwise construction was required due to the sequential behavior of the control systems as it was expected that safety property violation will occur after execution of a particular trajectory or sequence of data in the input domain. The test strategy was applied to a steam boiler case study where the target safety property was the non-explosion of the boiler. Along with the objective of violating a safety property, there was a set of dangerous situations of interest when exploring progressive evolution towards property violation. So the objective was not only violation of a target property but also to reach a dangerous situation. For the steam boiler case study, there could be ten possible safety property violations and three dangerous situations. The solution space in this case was divided into several subsets of smaller sizes. So different classes of test sequences were independently searched. For each class, the objective was defined into sub-objectives corresponding to either safety property violation or the achievement of a dangerous situation. The overall cost was the minimum value of the different sub-objective cost functions. The efficiency of using SA was analyzed in comparison with random sampling. The first experiment showed that random sampling found test sequences that fulfilled main objective more quickly than the approach using SA. Therefore a revised SA was used in which the acceptance probability was adjusted to allow for significant moves in case of no cost improvement. The revised version of SA offered significant improvement over the basic version of SA, while in comparison with random sampling a slight improvement was observed both in terms of total number of iterations and successful search. The results of the study confirmed the usefulness of stepwise construction of test scenarios, but in terms of efficiency of SA algorithm, the cost effectiveness as compared with random sampling remains questionable.

In [24, 206], an evolutionary testing approach using genetic algorithms was presented for structural and functional sequence testing. For complex dynamic system like car control systems, long input sequences were necessary to simulate these systems. At the same time, one of the most important aims was to generate a compact description for the input sequences, containing as few elements as possible but having enough variety to stimulate the system under test as much as necessary. In order to have a compact description of input sequences for a car control system, the long input sequence was divided into sections. Each section had a base signal having signal type, amplitude and length of section as variables. These variables had bounds within which the optimization generated solutions. The output sequences generated by simulating the car control system were to be evaluated against a fitness function, which was defined according to the problem at hand. For example, in case of a car control system, the fitness function checked for violations of signal amplitude boundaries. The applied fitness function consisted of two levels, which differentiated quality between multiple output signals violating the defined boundaries. An objective value of $-1$ indicated a severe violation while for less severe violations, the closeness of the maximal value to the defined boundary was calculated. The results of the experiment performed on a car control system showed that the optimization continually found better values and ultimately a serious violation was detected.

A summary of results of applying metaheuristics for safety testing is given in Table 5.9.

## 5.4 Discussion and areas for future research

The body of knowledge into the use of metaheuristic search techniques for verifying the temporal correctness is geared towards real-time embedded systems. For these systems, temporal correctness must be verified along with the logical correctness. The fact that there is a lack of support for dynamic testing of real-time system for temporal correctness caused the research community to take advantage of metaheuristic search

Table 5.9: Summary of results applying metaheuristics for safety testing. The last column on the right covers any issues such as constraints, limitations and highlights. (GA is short for Genetic Algorithm and SA is short for Simulated Annealing.)

| Article | Applied meta-heuristic | Fitness function used | Limitations and highlights |
|---|---|---|---|
| Tracey et al. 1999 [246] | GA and SA | Evaluation of different branch predicates with zero cost if the safety property evaluates to false and positive cost otherwise | **Experimentation** on small scale problems, thus results are preliminary. **Instrumentation** of the test objects is a challenge in the scalability of the technique. |
| Kaddour et al. 2003 [2] | SA | Cost related to the violation of the safety property and achievement of dangerous situation | **Using** SA, many trials are necessary for investigating alternative design choices and calibrating the corresponding parameters. **More** experimentation is required to confirm the efficiency applying revised SA algorithm. |
| Baresel et al. 2003 [24], and Pohlheim et al. 2005 [206] | GA | Problem-specific fitness function measuring different properties e.g. signal amplitude boundaries | **The** input sequences must be long enough and should have the right attributes to stimulate the system. **The** output sequence must be evaluated according to the problem under investigation. |

techniques. It is possible to differentiate the temporal testing research into two dimensions. One of them focuses on violation of timing constraints due to input values and most of the temporal testing research follows this dimension. The other dimension, which is the one taken by Briand et al. in [36] analyses task architectures and consider seeding times of events triggering tasks and tasks' synchronization, i.e. Briand's et al. study does not consider tasks in isolation. Both approaches to temporal verification are complementary.

The performance outcome information from studies related to execution time are given in Table 5.10. It is fairly evident from the table that GA consistently outperforms random and statistical testing in wide variety of situations, producing comparatively longer execution times faster and also finding new bounds on BCET. GAs were also able to perform better than human testers and on occasions where it failed to do so may be attributed to the complexity of the test objects inhibiting evolutionary testability. With respect to comparison with static analysis, GA performed comparably well and both techniques are shown to bound the actual execution times from opposite ends.

For execution time, genetic algorithms are used as the metaheuristic in vast majority of cases (14 out of 15 papers), while SA finds application in one of the studies. The preference of using genetic algorithms over SA can be attributed to the very nature of search mechanism inherent to genetic algorithms. Since a genetic algorithm maintains a population of possible solutions, it has a better chance of locating global optimum as compared to SA which proceed one solution at a time. Also due to the fact that temporal behavior of real-time systems always results in complicated multi-dimensional search space with many plateaus and discontinuities, genetic algorithms are suitable since

Table 5.10: Key evaluation information and outcomes of execution time studies. (GA is short for Genetic Algorithm, EGA is short for Extended GA, while SA is short for Simulated Annealing)

| Article and Metaheuristic | Method of evaluation | Test objects | Performance factor evaluated | Outcomes of the experiment |
|---|---|---|---|---|
| Wegener et al. 1996 [252], GA | Comparison with statistical and systematic testing | Simple C function | Finding WCET/BCET and number of generations | The longest execution time was found very fast and a new shortest execution time was found, not previously discovered by statistical and systematic testing |
| Alander et al. 1997 [14], GA | Comparison with random testing | Relay software simulator | Finding processing time extremes | GA generated input data with longer response times |
| Wegener et al. 1997 [255], GA | Comparison with statistical and systematic testing | 5 programs with up to 1511 LoC and 843 integer input parameters | Finding WCET/BCET and the number of tests required | GA found more extreme times although on occasions required more tests to do so |
| Wegener and Grochtmann 1998 [253], GA | Comparison with random testing | 8 programs with up to 1511 LoC and 5000 input parameters | Finding WCET/BCET and the number of tests required | GA always obtained better results as compared with random testing |
| O'Sullivan et al. 1998 [203], GA | Comparison with different termination criteria i.e. limiting the number of generations, time spent on test and examining the fitness evolution | An algorithm from automotive electronics | Convergence analysis of various termination criteria | Cluster analysis turned out to be a more powerful termination criterion which allowed quick location of local optima |
| Tracey et al. 1998 [245], SA | Comparison in terms of size of parameter space to search | 4 programs (conditional blocks, simple loop, binary integer square root and insertion sort) | Finding WCET | SA successfully executed a worst case path and was more effective with larger, complex parameter space |
| Pohlheim and Wegener 1999 [207], EGA | Comparison with systematic testing | Modules from a motor control system | Finding maximum execution times | GAs found longer execution times for all the given modules |
| Mueller and Wegener 1998 [185], GA | Comparison with static analysis | 5 experiments with industrial and reference applications | Finding WCET/BCET | Use of evolutionary testing and static analysis bounded the actual execution times from opposite ends and were complementary |
| Puschner and Nossal 1998 [211], GA | Comparison with static analysis and random testing | 7 programs with diverse execution time characteristics | Finding WCET | For large input data space, GA outperformed random method. In comparison with static analysis, the performance of GA is comparable |
| Wegener et al. 2000 [254], GA | Comparison with execution time found with developers test | An engine control system with 6 time-critical tasks | Finding WCET | Longer execution times were found with the evolutionary test than with the developer tests |
| Groß 2003 [92], GA | Random test case generation and performance of an experienced human tester | 15 example test programs | Finding WCET | Evolutionary testing generated more worst-case times as compared with random testing. Also for only 4 out of 15 test objects the human tester was more successful |
| Groß 2001 [91], GA | None | 21 test objects of varying input size | Evolutionary testability | The prediction system forecasted evolutionary testability with almost 90 |
| Groß et al. 2000 [94], GA | None | 22 test objects with varying input sizes | Evolutionary testability | Evolutionary testability and complexity of test objects was found to be interrelated |
| Tlili et al. 2006 [243], EGA | Standard evolutionary real-time testing with random initial population and no search space restriction | 12 test objects with varying cyclomatic complexity | Finding longer execution times | Using range restriction and seeding initial population with data achieving high structural branch coverage, longer execution times were found for most of the test objects except for two |
| Briand et al. 2005 [36], GA | None | Two case studies, one of researchers own scenarios and the second consisted of an actual real time system | Maximizing critical deadline misses | It was possible to identify seeding times such that small errors in the execution time estimates could lead to missed deadlines |

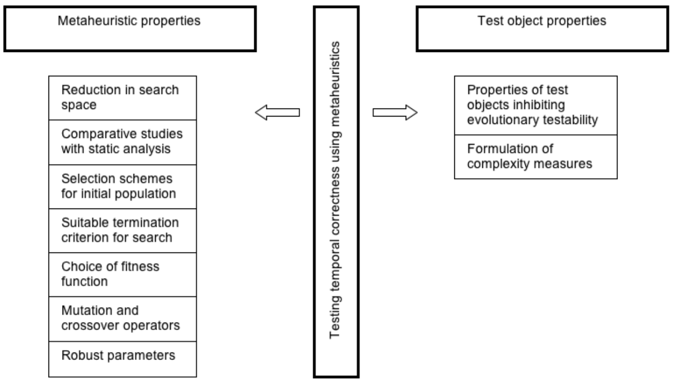

Metaheuristic properties
Reduction in search space
Comparative studies with static analysis
Selection schemes for initial population
Suitable termination criterion for search
Choice of fitness function
Mutation and crossover operators
Robust parameters
Testing temporal correctness using metaheuristics
Test object properties
Properties of test objects inhibiting evolutionary testability
Formulation of complexity measures

Figure 5.4: Two ways to analyze temporal testing research using metaheuristics.

they perform well for problems involving large number of variables and complex input domains.

There is another way of analyzing the body of knowledge into the use of metaheuristics for testing temporal correctness; which is in terms of (1) properties associated with the metaheuristic itself and (2) properties related to the SUT (test object). In terms of metaheuristic, the research focuses on improving multiple issues: reduction in search space, comparative studies with static analysis, selection schemes for initial population of genetic algorithm, evaluation of suitable termination criterion for search, choice of fitness function, mutation and crossover operators and search for robust and appropriate parameters. In terms of test objects, the research focuses on properties of test objects inhibiting evolutionary testability and formulation of complexity measures for predicting evolutionary testability (Figure 5.4).

The fact that seeding the initial population of an evolutionary algorithm with test data having high structural coverage has had better results, it would be interesting to design a fitness function that takes into account structural properties of individuals. Then it will be possible not only to reward individuals on the basis of execution time but also on their ability to execute complex parts of the source code. Similarly, there are different types of structural coverage criteria, which can be used to seed initial populations and might prove helpful in the design of a fitness function that takes into account such a structural coverage criterion.

In terms of reliable termination criterion for evolutionary testing, use of cluster analysis is found to be useful over other termination criteria, e.g. number of genera-

Table 5.11: Key evaluation information and outcomes of QoS studies. (GA is short for Genetic Algorithm)

| Article and Meta-heuristic | Method of evaluation | Test objects | Performance factor evaluated | Outcomes of the experiment |
|---|---|---|---|---|
| Canfora et al. 2005 [51], GA | Linear integer programming | A workflow containing 18 invocations of 8 distinct abstract services | Convergence times of integer programming and GA for the same achieved solution | When the number of concrete services available for each abstract service is large, GA should be preferred instead of integer programming. On the other hand, whenever the number of concrete services available is limited, integer programming is to be preferred |
| Di Penta et al. 2007 [69], GA | Random search | Audio processing workflow and a service for chart generation | Violation of QoS constraint and time required to converge to a solution | The new approach outperformed random search and successfully violated QoS constraints |

tions and examination of fitness evolution. However, to the authors' knowledge, the use of cluster analysis as a termination criterion is used in only one study [203]. Cluster analysis information can be used to change the search strategy in a way that escapes local optima and helps exploring more feasible areas of the search space. Also the performance of evolutionary algorithms can be made much better by using robust parameters. The search for these parameters is still on. Similarly, variations of existing algorithms e.g. like using extended evolutionary algorithms might give interesting insights into the performance of evolutionary testing.

We also gathered studies regarding application of metaheuristic search techniques for quality of service aware composition and violation of service level agreements (SLAs) between the integrator and the end user. Genetic algorithms have been applied to tackle the QoS-aware composition problem as well as generation of inputs causing SLA violations. Table 5.11 show that in comparison with linear integer programming and random search, genetic algorithms were more successful in meeting QoS constraints. We also infer that the testing of service-oriented systems has inherently several issues. These include testability problems due to lack of observability of service code and structure, integration testing issues due to the use of late-binding mechanisms, lack of control and involved cost of testing [69]. These issues raises the need for adequate testing strategies and approaches tailored for service-oriented systems. In terms of QoS, different attributes are of interest and are competing e.g. cost, response time, availability and reliability. These attributes need to be computed for workflow constructs. Empirical evidence has to be gathered for a thorough comparison of genetic algorithms with non-linear integer programming for QoS-aware service composition. Also network configurations and server load, being one of the factors causing SLA violations, is to be accounted for generating test data violating the SLA. Buffer overflow attacks compromises the security of applications. The attacker needs three requirements for a successful exploit, (i) a vulnerable program within the target system (ii) information of the size of memory reference necessary to cause the overflow and (iii) the correct placement of a suitable exploit to make use of the overflow when it occurs [131]. In order to guide the interpretation of findings, performance outcomes are summarized in Table 5.12, in addition to Table 5.5. The range of studies offers variations with respect to the main theme, although all have the common goal of addressing security testing. Therefore, we see studies making use of metaheuristic search techniques to create a range of successful attacks to evade common intrusion detection systems as well as to identify buffer overflows. For detecting buffer overflows, the attacker's arbitrary code needs modification to increase the success chances of creating a malicious buffer. In case of hacker scripts generation, the goals for the hacker script generation need to be defined. The use of metaheuristic search techniques includes grammatical evolution, linear genetic programming, genetic algorithm and particle swarm optimization. Devising a useful fitness function is the focus of majority of the studies which highlights the difficulty in instrumenting security issues as an appropriate search metric. Most of the fitness functions are based on the ability of the attack to fulfill the conditions necessary for a successful exploit. This has resulted in studies comparing the performances of different fitness functions rather than comparing with traditional approaches to security testing. This indicates that most of the studies into use of metaheuristics for this domain is largely exploratory and no trends are observable that can be generalized. Due to this we see authors experimenting with simple and small applications on a limited scale. The scalability of these approaches, with larger data sets and greater number of trials, is an interesting area of future research. The work of Kayacik et al. [132, 131, 133] is notable as they move towards a general framework for attack generation based on the evolution of system call sequences. Also the co-evolution of attacker-detector pairs (as pointed out by Kayacik et al. [133]) that provides the opportunity to actually preempt new attack behaviors before they are encountered, formulation of techniques (like static analysis and program slicing) to reduce the search space for the evolutionary algorithm, hybridization of GA and PSO algorithm for effective searching and creation of an interactive tool for vulnerability assessment based on evolutionary computation for the end users provides further future extensions.

In the area of usability, we found the application of metaheuristic search techniques for constructing covering arrays. Being a qualitative attribute, usability possesses different interpretations. However, we classify studies reporting construction of covering arrays under usability as they relate to a form of interaction testing covering $t$-way user interactions whereby each test case exposes different areas of a user interface. The research for construction of covering arrays for software testing have dual focus of finding new techniques to produce smaller covering arrays and to produce them in reasonable amount of time. As expected, a trade-off must be achieved between computational power and size of resulting test suites. The extent of evidence related to applying metaheuristics for finding better bounds on covering arrays suggest that metaheuristics are very successful for smaller parameter sets. For larger parameter sets, the heuristic algorithms run slowly due to the large amount of memory required (to store information). Therefore, execution time is a known barrier in finding more results using metaheuristic search algorithms. One obvious way to achieve efficient memory management is to reuse the previously calculated $t$-combinations by storing them in some form of a temporary memory. Also as mentioned earlier, the size of the test suite is not known a priori, many sizes must be tested for obtaining a good bound on a $t$-way test set of given size.

It is also worth mentioning that finding optimal parameters for effectively using metaheuristics require many trials, moreover it is difficult to be sure whether the upper bounds produced are optimal or not because further improvements on bounds can take place if given more computing time. In addition to Table 5.7, we summarize key evaluation information and outcomes in Table 5.13. We gather that a range of metaheuristics have been applied for constructing covering arrays. This includes SA, TS, GA and ant colony optimization. We find SA and TS to be widely applicable search techniques, particularly SA being applied to generate smaller sized test suites. Out of seven primary studies, five report using SA while three use TS, either being used independently or in combination with other search techniques and algebraic constructions. We infer from Table 5.13, SA consistently performs better than GA, HC, ACA and TS in terms of size of resulting test sets. The performance of SA can further be improved by integrating it with the use of algebraic constructions. Another possibility is to begin with a greedy algorithm (like TCG) and then make a transition to heuristic search after meeting a certain condition [55]. GA has been applied in two studies with contradictory results and hence requires further experimentation. An interesting area is to explore the use of ant colony optimization to generalize initial results given by [225]. In terms of construction of variable strength covering arrays, there is a potential for further research for finding the best approach for variable strength test suite.

Safety testing is used to test safety critical systems that have to satisfy the safety constraints in addition to satisfying the functional specification. We take safety in terms of dangerous conditions, which may contribute to an accident. There are two approaches for achieving verification of safety properties: dynamic testing and static analysis. Static analysis does not require execution of the safety-critical system while dynamic testing executes the system in a suitable environment with test data generated to test safety properties. Both static analysis and dynamic testing are complementary approaches; in many cases the results of static analysis are used to give criteria for dynamic testing (as in [246]). The available primary studies discussing testing of safety

Table 5.12: Key evaluation information and outcomes of security studies. (GA is short for Genetic Algorithm, GE is short for Grammatical Evolution, while PSO is short for Particle Swarm Optimization)

| Article and Metaheuristic | Method of evaluation | Test objects | Performance factor evaluated | Outcomes of the experiment |
|---|---|---|---|---|
| Dozier et al. 2004 [75], GA and PSO | Comparison of steady state GA and six variants of PSO | 1998 MIT Lincoln Lab Data | To discover holes (Type II errors/false negatives) | GA outperformed all of the swarms with respect to number of distinct vulnerabilities discovered |
| Kayacik et al. 2005 [132], GE (basic, niching and niching & NoOP minimization) | Detection (or not) of each exploit through the Snort misuse detection system | A simple (generic) vulnerable application performing a data copy without checking the internal buffer size | The number of alerts that Snort generates when attacks are executed | The results from three variants of GE were comparable |
| Budynek et al. 2005 [44], GA | Results from a log analyzer | Automatic generation of computer hacker scripts (a sequence of Unix commands) | Evidence collection from the logs | Various top scoring scripts were obtained |
| Grosso et al. 2005 [96], GA | Comparison of three different fitness functions namely vulnerable coverage fitness, nesting fitness and buffer boundary fitness | A white-noise generator and a function contained in the ftp client | Modified $t$-test to compare fitness values of three fitness functions across the two test objects | A fitness function accounting for the distance from the buffer boundaries outperformed fitness function not using the same factor |
| Grosso et al. 2007 [95], GA | The fitness that does not use dynamic weighing | Two different sets of C applications | A comparison of fitness values of competing fitness cases in terms of number of generations | The new fitness function outperformed the comparable ones |
| Kayacik et al. 2006 [131], Linear GP | Evaluation of a new approach | Three experiments with different data sets | Avoidance of attack detection by Snort, the network based intrusion detection system | The evolved attacks discovered different ways of attaining sub-goals associated with building buffer overflow attacks |
| Kayacik et al. 2007 [133], Linear GP | Compares two fitness functions (incremental and concurrent) in detecting anomaly rate | Instruction set consisting of most frequently occurring system calls | Measurement of alarm rate indicating evasion of Stride, the anomaly host based detection system | Reduction of anomaly rate from $\sim 65\%$ to $\sim 2.7\%$ |

Table 5.13: Key evaluation information and outcomes of usability studies. (GA is short for Genetic Algorithm, TS is short for Tabu Search, SA is short for Simulated Annealing, HC is short for Hill Climbing, PSO is short for Particle Swarm Optimization while, ACA is short for Ant Colony Algorithm)

| Article and Metaheuristic | Method of evaluation | Test objects | Performance factor evaluated | Outcomes of the experiment |
|---|---|---|---|---|
| Stardom 2001 [233], SA, TS and GA | Comparison of SA, TS and GA | Different sizes of covering arrays, CA(13,11:1), CA(9,7:1), CA(7,6:1) | Three tests to find the best arrays possible in the shortest amount of time | GA was ineffective at finding quality arrays when compared with TS and SA. SA in general was found to be very useful for finding covering arrays of various sizes while when the size of an arrays neighborhood was smaller, TS was able to find much better arrays |
| Cohen et al. 2003 [55], SA and HC | Comparison of SA, HC and greedy methods (AETG, TCG) | Different sizes of mixed covering arrays and fixed covering arrays | Number of test cases in a test suite and time required to obtain them | HC and SA improved on bounds given by AETG and TCG, SA consistently performed well or better than HC |
| Cohen et al. 2003 [56], SA | Presentation of results for a new combinatorial object (variable strength covering array) | Minimum, maximum and average sizes of different variable strength covering arrays | Minimum, maximum and average sizes of variable strength covering arrays after 10 runs of SA | Variable strength covering arrays of different sizes |
| Nurmela 2003 [196], TS | Comparison with best known upper bounds | Upper bounds on $g_2(Z_n^q)$ for small $q$ and $n$ | Size of the covering array | The search algorithm worked best for $t$=2 and $q$=3 |
| Cohen et al. 2003 [174], SA | Comparisons with strength three covering arrays | Several bounds for strength three covering arrays | Size of strength three covering array | Combination of combinatorial construction and SA presented new bounds for some strength 3 covering arrays |
| Bryce et al. 2007 [42], TS, SA, HC | Comparisons among TS, SA and HC | Two inputs with factors having equal number of levels and two inputs having mixed number of levels | Rate of $t$-tuple coverage | SA had the fastest rate of $t$-tuple coverage |
| Toshiaki et al. 2004 [225], ACA, GA | Comparisons with AETG, IPO and SA algorithms for the cases $t$=2 and $t$=3 | Covering arrays and mixed covering arrays of strength 2 and 3 | Size of resulting test sets and amount of time required for generation | For $t$=2, GA and ACA performed comparable to AETG. For $t$=3, GA and ACA outperformed AETG. SA outperformed GA and ACA with respect to size of the resulting test sets |

properties can be differentiated into two themes. One is the case where generation of separate inputs is discussed to test the safety property while the other case discusses generation of sequence of inputs. The performance outcomes for studies related to safety testing are given in Table 5.14. The studies show that SA and GAs are applied

Table 5.14: Key evaluation information and outcomes of safety studies. (GA is short for Genetic Algorithm, while SA is short for Simulated Annealing)

| Article and Metaheuristic | Method of evaluation | Test objects | Performance factor evaluated | Outcomes of the experiment |
|---|---|---|---|---|
| Tracey et al. 1999 [246], GA and SA | Comparison of safety conditions obtained from software fault tree analysis and functional specification | Small size functions used in the pre-proof step | Finding test data that causes an implementation to violate a safety property | The approach might be useful not only for safety verification but also for integration with fault injection, testing for exception conditions and testing for safe component reuse |
| Kaddour et al. 2003 [2], SA | Effectiveness in terms of finding appropriate test sequences and efficiency in terms of comparing the speed of SA with random sampling | Non-explosion of the steam boiler | Finding the test sequence that lead to either an explosion or a dangerous situation | Both random search and SA were effective while random search was more efficient |
| Baresel et al. 2003 [24], Pohlheim et al. 2005 [206], GA | None | Dynamic car control system | Violation of defined requirements for output sequence generated by the simulation of the dynamic system | It was possible to generate real-world input sequences causing violations of a given safety requirement |

in the context of safety testing, GA being more successfully applied. However, the results suggest a need for further experimentation in terms of investigating alternative design choices and calibration of algorithmic parameters. This is desirable especially when the extensions to the basic approaches of safety testing can be applied to fault injection, testing for exception conditions and testing for safe component reuse and integration [246]. In terms of alternate design choices, we see in [2] that the efficiency of SA is dependent on the initial solution because when no cost improvement is observed, the search does not allow moves larger than those authorized by the neighborhood function. Therefore, improvement in the efficiency of the SA algorithm can be achieved by searching elsewhere then the neighborhood of the current solution if no cost improvement is made. Further experimentation is also desirable in terms of safety testing real-world applications, however, the instrumentation required for testing of safety conditions is a challenge with respect to the scalability of the technique. Also, since the design choices are highly dependent on the nature of the safety critical system, it is consequently important to include more problem-specific knowledge into the representation of solutions and design of fitness function.

We can infer from this review that search-based testing is poorly represented in the testing of non-functional properties. While search-based software engineering might be transitioning from early optimistic results to more in-depth understanding of the associated problems [103], search-based testing of non-functional properties is still adhoc and largely exploratory. The main reasons that can be attributed to this trend are the difficulties associated with instrumenting non-functional properties as fitness functions and also difficulties in generalizing search-based testing of non-functional properties on a broader scale due to strictly domain specific nature of existing studies. With the majority of studies in search-based software testing applied to functional testing, the use of metaheuristic search techniques for testing non-functional properties is rather limited and, with exception to execution time studies, are very problem specific. We believe that it is important, in order to develop the currently emerging field of search-based software testing, to analyze the applicability of search techniques in testing of diverse non-functional properties which can then trigger the second phase of exploration requiring a deeper understanding of problem and solution characteristics [103].

## 5.5 Validity threats

There can be different threats to the validity of study results.

*Conclusion validity* refers to the statistically significant relationship between the treatment and the outcome [258]. One possible threat to conclusion validity is biasness in applying quality assessment and data extraction. In order to minimize this threat, we explicitly define the inclusion and exclusion criteria, which we believe is detailed enough to provide an assessment of how we reached the final set of papers for analysis. With respect to the quality assessment, we wanted to be as inclusive as possible, so we resorted to a binary 'yes' or 'no' scale rather than assigning any scores. We made sure to a large extent *include* instead of *exclude* references, hence making sure not to place, by mistake, any relevant contributions in the 'no' category. To assess the consistency of data extraction, a small sample of primary studies were used to extract data for the second time.

*Internal validity* refers to a causal relationship between treatment and outcome [258]. One threat to internal validity arises from unpublished research that had undesired outcomes or proprietary literature that is not made available. It is difficult to find such grey literature; however we acknowledge that inclusion of such literature would have contributed in increasing internal validity.

*Construct validity* is concerned with the relationship between the theory and application [258]. One possible threat to construct validity is exclusion of relevant studies. In order to minimize this threat, we defined a rigorous search strategy (Subsect. 5.2.2), which included two phases, to ultimately protect us against threats to construct validity.

*External validity* is concerned with the generalization of results outside the scope of the study [258]. We can relate it to the degree to which the primary studies are representative of the overall goal of the review. We believe that our review protocol helped us achieve a representative set of studies to a greater extent. During the course of scanning references, the authors also came across two studies by Schultz et al. [220, 221], applying evolutionary algorithms for the robustness testing of autonomous vehicle controllers. We do not include these two studies in our analysis since these studies were published in 1992 and 1995, which are outside the time span (1996–2007) of this review. Furthermore, we did not anticipate finding other relevant studies outside the time span of 1996–2007 as previous relevant surveys supports such a choice of time span.

## 5.6 Conclusions

This systematic review investigated the use of metaheuristic search techniques for testing non-functional properties of the SUT. The 35 primary studies are distributed among execution time (15 papers), quality of service (2 papers), safety (4 papers), security (7 papers) and usability (7 papers). While scanning references, we also found two papers relating to robustness testing of autonomous vehicle controllers [220, 221] but we do not include these two papers in our review as they were outside the time span of our search (1996 to 2007).

Within execution time testing, genetic algorithms finds application in 14 out of 15 papers. The research trend within execution time testing is more towards violation of timing constraints due to input values; however, the paper by Briand et al. [36] provides another research approach that analyzes the task architectures and consider seeding times of events triggering tasks and tasks' synchronization. In terms of use of fitness function, we find three variations; the execution time measured in CPU clock cycles, coverage of code annotations inserted along shortest and longest algorithmic execution paths and the fitness function based on the difference between execution's deadline and execution's actual completion. The challenges identified include dealing with potential probe effects due to instrumentation and uncertainty about global optimum, finding appropriate and robust search parameters and a having a suitable termination criteria of search.

Within quality of service (QoS), the two papers apply genetic algorithms to determine the set of service concretizations that lead to QoS constraint satisfaction and to generate combinations of bindings and inputs causing violations of service level agreements. One of the papers uses a fitness function based on the maximization of desired QoS attributes while minimizing others and includes the possibility of having static or dynamic fitness function. The other paper uses a fitness function that combines distance-based fitness with a fitness guiding the coverage of target statements. The challenges include the need to compute QoS attributes of component services for workflow constructs and to deal with the possibility of deviation of fitness calculation due to workflow instrumentation.

In security testing; genetic algorithms, linear genetic programming, grammatical evolution and particle swarm optimization have been applied. The applied fitness functions used different representations for the completion of conditions leading to successful exploits. Modifications to the attacker's arbitrary code and finding appropriate goals for hacker script generation are identified as the challenges for security testing.

Within usability testing, metaheuristic search techniques are applied to find better bounds on covering arrays. A variety of metaheuristic search techniques are applied including SA, TS, GA and ant colony algorithms. The fitness function used is the number of uncovered $t$-subsets. Execution time is a major challenge in this case as for large parameter sets, the metaheuristic algorithms run slowly due to the large amount of memory required to store information.

In safety testing, we find two research directions to test safety properties of the SUT. One makes use of generation of separate inputs to test the safety property, while the other uses a sequence of inputs. Simulated annealing and genetic algorithms are the used metaheuristics and the fitness function takes into account the violation of various safety properties. The incorporation of problem specific knowledge into the representation of solution and design of fitness function presents a challenge for the application of metaheuristic search techniques to test safety properties.

We believe that there is still plenty of potential for automating non-functional testing using search-based techniques and we expect that studies involving NFSBST will increase in the following years. The results of our systematic review also indicate that the current body of knowledge concerning search-based software testing does not report studies on many of the other non-functional properties. On the other hand, there is a need to extend the early optimistic results of applying NFSBST to larger real world systems, thus moving towards a generalization of results.

# Chapter 6

# Summary and conclusions

## 6.1 Summary

The research questions (Section 1.3) in this thesis are targeted towards two activities within software V&V: Software predictive modeling and software testing. The first two specific research questions (RQ1.1–1.2) are focused on empirical evaluation of symbolic regression using genetic programming as a predictive modeling technique. The third specific research question (RQ2.1) reviews the state of research within testing of non-functional system properties using search-based techniques.

Chapter 2 comprises of an initial investigation into the predictive capabilities of applying symbolic regression using genetic programming. The comparative evaluation with traditional software reliability growth models shows that symbolic regression using genetic programming has a potential to be a valid fault prediction technique. Using three measures of model validity, 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 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 a positive result in favor of GP models.

Chapter 3 carries forward the early positive results of using symbolic regression application of genetic programming. The empirical investigation this time is into cross-release prediction of fault-count data from large and complex industrial and open-source software. The results are evaluated both quantitatively and qualitatively, while the comparisons are done with both machine-learning and traditional approaches to fault prediction. The results show that, quantitatively, symbolic regression using genetic programming is at least as competitive as other techniques for cross-release fault prediction. Qualitatively, symbolic regression using genetic programming scores better for transparency of resulting solutions and generality, in comparison with comparative techniques. On the other hand, ease of configuration is not a strength for symbolic regression using genetic programming.

Chapter 4 consolidates the existing evidence in the software engineering literature that comparatively evaluates the symbolic regression application of genetic programming with other techniques. The results of this study provide evidence in support of symbolic regression using genetic programming for software quality classification, software fault prediction and software reliability growth modeling in comparison with regression/machine learning techniques and other models.

Chapter 5 focuses on conducting a systematic review of search-based approaches for testing non-functional system properties. The results of the review show that meta-heuristic search techniques have been applied for non-functional testing of execution time, quality of service, security, usability and safety. The results further show the applicability of different search techniques including simulated annealing, tabu search, genetic algorithms, ant colony methods, grammatical evolution, genetic programming and swarm intelligence methods.

## 6.2 Conclusions

What we can conclude from the studies in this thesis is that a search-based technique like genetic programming can be applied for software fault predictions (as investigated in Chapters 2, 3 and 4). This is in addition to the existing literature evidence showing the application of different search-based techniques for testing non-functional system properties (as investigated in Chapter 5).

Another important conclusion that can be drawn from the studies on software fault prediction in this thesis is that while use of symbolic regression application of genetic programming had benefits (as outlined in Section 1.1), there were also shortcomings. These shortcomings were concerned with difficulties in configuring the genetic programming algorithm, which had different parameters to configure, and with the complexity of the resulting solutions.

More specific conclusions, which can be drawn from this thesis, are given below:

1. The evolutionary search mechanism of symbolic regression using genetic programming was suitable for predicting the future software reliability in terms of number of faults (Section 2.4).

2. 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 genetic programming (Section 2.5).

3. Using the same data as in bullet 2 above, the comparative evaluation of models from symbolic regression application of genetic programming and three traditional software reliability growth models showed that 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 genetic programming were at least competitive to traditional software reliability growth models (Section 2.6).

4. The quantitative evaluation of models built using symbolic regression application of genetic programming showed that for cross-release fault predictions, they were at least as competitive as other machine learning and traditional models (Section 3.7).

5. There is 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 genetic programming 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 genetic programming (Sections 3.9 and 4.4).

6. While there was evidence in software engineering literature in support of using symbolic regression application of genetic programming for software quality classification, software fault prediction and software reliability growth modeling; we were inconclusive for software cost/effort/size estimation (Section 4.4).

7. The results of the systematic review in Chapter 5 suggested that execution time is the most suitable non-functional system property for applying search-based software testing. In terms of search techniques, genetic algorithm was the mostly used search technique.

This thesis has given indications that search-based techniques are applicable for activities within software verification and validation. The evolutionary search mechanism of symbolic regression using genetic programming has shown some encouraging results for predictive modeling of fault data; that is, in addition to the literature evidence where search-based techniques have been applied successfully for testing non-functional system properties.

## 6.3 Future research

Through empirical investigations and literature reviews 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. These research opportunities are grouped into different themes and given below.

- Industrial
  - Apply and evaluate the early positive results of using genetic programming for software fault predictions in an on-going industrial project.
- Scope expansion
  - Empirically evaluate the effectiveness of using genetic programming in prediction across other phases of the software development life cycle, such as maintenance task effort.
  - Investigate the accuracy of predictions from the genetic programming algorithm early on in a software development life cycle.
  - Evaluate the use of other search-based approaches for predictions, such as particle swarm optimization, artificial immune programming and gene expression programming.
- Algorithmic enhancements
  - Empirically evaluate the use of different fitness functions to better guide search of feasible solutions for the genetic programming algorithm.
  - Investigate the mechanisms of finding compact and less complex genetic programming solutions.
  - Investigate the potential of saving the state information during genetic programming evolved solutions so as to enhance the predictive accuracy on time-series nature of data.
- Design enhancements
  - Assess the impact of different lengths of the training data and different cross-validation schemes on the effectiveness of the predictions from the genetic programming algorithm.

- Investigate the potential of adaptive parameter control during a genetic programming run to ease parameter tuning for the genetic programming algorithm.
- Evaluate the adaptive capability of the genetic programming algorithm for different sets of independent software metrics.
- Evaluate the effectiveness of genetic programming predictions at finer levels of detail by collecting different metrics at the code and/or module level.
- Increase the number of comparisons with other competitive approaches, such as Bayesian networks [80] and MARS [38].
- Investigate the possibility of having an ensemble of techniques for software fault prediction at hand for use by software engineers working in real-world projects.

The above mentioned opportunities indicate that there are future research opportunities along multiple dimensions. While there are opportunities to seek algorithmic improvements within a particular search-based technique like genetic programming, there are also open opportunities to apply other search-based techniques. Furthermore, there are opportunities to evaluate existing applications of search-based techniques in a particular domain and, especially, in real-world settings, i.e. in ongoing projects in industry.

# References

[1] Institute of Electrical and Electronics Engineers recommended practice for software requirements specifications. Technical Report 830–1998, Institute of Electrical and Electronics Engineers, Inc., 1998.

[2] O. A.-Kaddour, P. Thévenod-Fosse, and H. Waeselynck. Property-oriented testing based on simulated annealing. `http://www.laas.fr/~francois/SVF/seminaires/inputs/02/olfapaper.pdf`, Last checked 05 Mar 2009.

[3] 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.

[4] 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.

[5] W. Adnan, M. Yaakob, R. Anas, and M. Tamjis. Artificial neural network for software reliability assessment. In *Proceedings of IEEE TENCON'00*, 2000.

[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. 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.

[8] 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.

[9] W. Afzal, R. Torkar, and R. Feldt. A systematic mapping study on non-functional search-based software testing. In *Proceedings of the 20th International Conference on Software Engineering & Knowledge Engineering (SEKE'08)*. Knowledge Systems Institute Graduate School, 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] 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.

[12] F. Akiyama. An example of software system debugging. *International Federation for Information Processing Congress*, 71(1):353–359, 1971.

[13] J. T. Alander. An indexed bibliography of genetic programming. Report Series no 94-1-GP, Department of Information Technology and Industrial Management, University of Vaasa, Finland, 1995. Last checked 13 Feb 2009.

[14] J. T. Alander, T. Mantere, G. Moghadampour, and J. Matila. Searching protection relay response time extremes using genetic algorithm-software quality by optimization. In *Proceedings of the 4th International Conference on Advances in Power System Control, Operation and Management (APSCOM'97)*, 1997.

[15] E. Alba and F. Chicano. Finding safety errors with ACO. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, pages 1066–1073, New York, NY, USA, 2007. ACM.

[16] 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.

[17] 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*, 2001.

[18] E. Alpaydin. *Introduction to machine learning*. The MIT Press, 2004.

[19] 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.

[20] F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, February 1973.

[21] 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.

[22] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The next release problem. *Information and Software Technology*, 43(14):883– 890, 2001.

[23] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic programming - An introduction*. Morgan Kaufmann Publishers, Inc., 1998.

[24] A. Baresel, H. Pohlheim, and S. Sadeghipour. Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms. In *Genetic and Evolutionary Computation—GECCO 2003*, volume 2724 of *Lecture Notes in Computer Science*, pages 2428–2441. Springer, 2003.

[25] 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.

[26] 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.

[27] B. Boehm and V. R. Basili. Software defect reduction top 10 list. *Computer*, 34(1):135–137, 2001.

[28] B. W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[29] S. Bouktif, G. Antoniol, E. Merlo, and M. Neteler. A novel approach to optimize clone refactoring activity. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, New York, NY, USA, 2006. ACM.

[30] S. Bouktif, H. Sahraoui, and G. Antoniol. Simulated annealing for improving software quality prediction. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, New York, NY, USA, 2006. ACM.

[31] 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.

[32] 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.

[33] L. 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.

[34] L. Briand, K. Emam, and S. Morasca. On the application of measurement theory in software engineering. ISERN-95-04.

[35] 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.

[36] L. C. Briand, Y. Labiche, and M. Shousha. Stress testing real-time systems with genetic algorithms. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*. ACM Press, 2005.

[37] L. C. Briand, Y. Labiche, and M. Shousha. Using genetic algorithms for early schedulability analysis and stress testing in real-time systems. *Genetic Programming and Evolvable Machines*, 7(2):145–170, 2006.

[38] L. C. Briand, W. L. Melo, and J. Wust. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering*, 28(7):706–720, 2002.

[39] S. Brocklehurst and B. Littlewood. Techniques for prediction analysis and recalibration. In *Handbook of software reliability engineering, Editor M. R. Lyu*, Hightstown, NJ, USA, 1996. McGraw-Hill, Inc.

[40] W. D. Brooks and R. W. Motley. Analysis of discrete software reliability models. Technical report, IBM FEDERAL SYSTEMS, 1980. ADA086334.

[41] R. C. Bryce. Automatic generation of high coverage usability tests. In *Extended Abstracts on Human Factors in Computing Systems (CHI'05)*, New York, NY, USA, 2005. ACM.

[42] R. C. Bryce and C. J. Colbourn. One-test-at-a-time heuristic search for interaction test suites. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, New York, NY, USA, 2007. ACM.

[43] 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.

[44] J. Budynek, E. Bonabeau, and B. Shargel. Evolving computer intrusion scripts for vulnerability assessment and log analysis. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*, New York, NY, USA, 2005. ACM.

[45] 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.

[46] 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.

[47] L. I. Burke. Introduction to artificial neural systems for pattern recognition. *Computers & Operations Research*, 18(2), 1991.

[48] 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.

[49] 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.

[50] K. Y. Cai, C. Wen, and M. Zhang. A novel approach to software reliability modeling. *Microelectronics and Reliability*, 33(15):2265–2267, 1993.

[51] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*. ACM, 2005.

[52] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346 – 7354, 2009.

[53] V. U. B. Challagulla, F. B. Bastani, I. Yen, and R. A. Paul. Empirical assessment of machine learning based software defect prediction techniques. In *WORDS '05: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 263–270, Washington, DC, USA, 2005. IEEE Computer Society.

[54] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7), 1997.

[55] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn. Constructing test suites for interaction testing. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*. IEEE Computer Society, 2003.

[56] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn, and J. S. Collofello. Variable strength interaction testing of components. In *Proceedings of the 27th Annual International Conference on Computer Software and Applications (COMPSAC'03)*. IEEE Computer Society, 2003.

[57] B. T. Compton and C. Withrow. Prediction and control of Ada software defects. *Journal of Systems and Software*, 12(3):199–207, 1990.

[58] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software engineering metrics and models*. Benjamin/Cummings, 1986.

[59] E. Costa, G. 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.

[60] 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.

[61] E. O. Costa, S. R. 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)*, pages 171–180, Washington, DC, USA, 2005. IEEE Computer Society.

[62] 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*. Swets en Zeitlinger B.V., 2003.

[63] J. W. Creswell. *Research design – Qualitative, quantitative and mixed method approaches*. Sage Publications, United Kingdom/India, second edition, 2003.

[64] Y. S. Dai, M. Xie, K. L. Poh, and B. Yang. Optimal testing-resource allocation with genetic algorithm for modular software systems. *Journal of Systems and Software*, 66(1):47–55, 2003.

[65] 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.

[66] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[67] 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.

[68] K. Derderian, R. M. Hierons, M. Harman, and Q. Guo. Input sequence generation for testing of communicating finite state machines (CFSMs). In *Proceedings of the 6th Annual Conference on Genetic and Evolutionary Computation (GECCO'04)*. ACM, 2004.

[69] M. Di Penta, G. Canfora, G. Esposito, V. Mazza, and M. Bruno. Search-based testing of service level agreements. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*. ACM Press, 2007.

[70] 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.

[71] J. J. Dolado. A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, 26(10), 2000.

[72] J. J. Dolado. On the problem of the software cost function. *Information and Software Technology*, 43(1):61 – 72, 2001.

[73] 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.

[74] 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*, 1999.

[75] G. V. Dozier, D. Brown, J. Hurley, and K. Cain. Vulnerability analysis of immunity-based intrusion detection systems using evolutionary hackers. In *Proceedings of the 6th Annual Conference on Genetic and Evolutionary Computation (GECCO'04)*. Springer, 2004.

[76] 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.

[77] T. Dybå, B. A. Kitchenham, and M. Jørgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):58–65, 2005.

[78] M. Evett, T. Khoshgoftar, P. d. Chien, and E. Allen. GP-based software quality prediction. In *Proceedings of the 3rd Annual Genetic Programming Conference*, 1998.

[79] W. Farr. SMERFS3 homepage, 2009. `http://www.slingcode.com/smerfs/downloads/`, Last checked 05 Mar 2009.

[80] N. Fenton, M. Neil, W. Marsh, P. Hearty, Ł. Radliński, and P. Krause. On the effectiveness of early life cycle defect prediction with bayesian nets. *Empirical Software Engineering*, 13(5):499–537, 2008.

[81] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999.

[82] 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.

[83] N. E. Fenton and S. L. Pfleeger. *Software metrics: A rigorous and practical approach*. Course Technology, Boston, MA, USA, 2nd edition, 1998.

[84] D. G. Firesmith. Common concepts underlying safety, security, and survivability engineering. Technical Report CMU/SEI-2003-TN-033, Software Engineering Institute, Pittsburgh, Pennsylvania, 2003.

[85] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering*, 29(11), 2003.

[86] J. E. Gaffney. Estimating the number of faults in code. *IEEE Transactions on Software Engineering*, 10(4):459–465, 1984.

[87] K. Gao and T. Khoshgoftaar. A comprehensive empirical study of count models for software fault prediction. *IEEE Transactions on Reliability*, 56(2), 2007.

[88] A. L. Goel. Software reliability models: Assumptions, limitations, and applicability. *IEEE Transactions on Software Engineering*, SE-11(12):1411–1423, 1985.

[89] 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.

[90] 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.

[91] H.-G. Groß. A prediction system for dynamic optimization-based execution time analysis. In *Proceedings of the 1st International Workshop on Software Engineering using Metaheuristic Innovative Algorithms (SEMINAL'01)*, Toronto, Canada, 2001.

[92] H.-G. Groß. An evaluation of dynamic, optimisation-based worst-case execution time analysis. In *Proceedings of the International Conference on Information Technology: Prospects and Challenges in the 21st Century (ITPC'03)*, Kathmandu, Nepal, 2003.

[93] H.-G. Groß, B. Jones, and D. Eyres. Evolutionary algorithms for the verification of execution time bounds for real-time software. In *IEE Colloquium on Applicable Modelling, Verification and Analysis Techniques for Real-Time Systems*, 1999.

[94] H.-G. Groß, B. F. Jones, and D. E. Eyres. Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems. *IEE Proceedings—Software*, 147(2):25–30, 2000.

[95] C. D. Grosso, G. Antoniol, E. Merlo, and P. Galinier. Detecting buffer overflow via automatic test input data generation. *Computers and Operations Research (COR) focused issue on search based software engineering*, 35(10):3125–3143.

[96] C. D. Grosso, G. Antoniol, M. Di Penta, P. Galinier, and E. Merlo. Improving network applications security: A new heuristic to generate stress testing data. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*, pages 1037–1043, New York, NY, USA, 2005. ACM.

[97] S. R. Gunn. Support vector machines for classification and regression. Technical report, School of electronics and computer science, University of Southampton, 1998.

[98] 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.

[99] 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.

[100] T. Gyimothy, 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.

[101] M. H. Halstead. *Elements of software science*. Elsevier, North-Holland, 1977.

[102] 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.

[103] 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.

[104] M. Harman and J. Clark. Metrics are fitness functions too. In *Proceedings of the 10th International Symposium on Software Metrics (METRICS'04)*. IEEE, 2004.

[105] M. Harman and B. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833 – 839, 2001.

[106] K. Henningsson. *A fault classification approach to software process improvement*. Blekinge Institute of Technology Licentiate Series No. 2005:03, Ronneby, Sweden, 2005.

[107] 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.

[108] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence*. MIT Press (reprinted 1992), Cambridge, MA, USA, 1975.

[109] M. Hollander and D. A. Wolfe. *Non-parametric statistical methods*. John Wiley and Sons, Inc., 1999.

[110] D. Hoskins, R. C. Turban, and C. J. Colbourn. Experimental designs in software engineering: D-optimal designs and covering arrays. In *Proceedings of the 2004 ACM workshop on Interdisciplinary Software Engineering Research (WISER'04)*, New York, NY, USA, 2004. ACM.

[111] The Institute of Electrical and Electronics Engineers, Inc., Los Alamitos, California. *Guide to the Software Engineering Body of Knowledge (SWEBOK ®)*, 2004.

[112] J. P. A. Ioannidis. Why most published research findings are false. *PLoS Medicine*, 2(8):696–701, August 2005.

[113] ISBSG. The International Software Benchmarking Standards Group Limited, 2009. Last checked 18 Mar 2009.

[114] ISO. International standard ISO/IEC 9126, information technology—product quality—part1: Quality model. Technical report, International Standard Organization, 2001.

[115] Z. Jelinski and P. B. Moranda. Software reliability research. In *Statistical Computer Performance Evaluation, Ed. W. Freiberger*, pages 465–497. Academic Press, New York, USA, 1972.

[116] C. Jones. The pragmatics of software process improvements, 1996. Software Process Newsletter of the Software Engineering Technical Council Newsletter, No. 5.

[117] M. Jørgensen. Experience with the accuracy of software maintenance task effort prediction models. *IEEE Transactions on Software Engineering*, 21(8):674–681, 1995.

[118] M. Jørgensen. BESTweb, a repository of software cost and effort estimation papers – Simula research laboratory, 2009. Last checked 07 Mar 2009.

[119] M. Jørgensen, T. Dybå, and B. Kitchenham. Teaching evidence-based software engineering to university students. In *Proceedings of the 11th IEEE International Symposium on Software Metrics (METRICS'05)*, 2005.

[120] M. Jørgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, 33(1):33–53, 2007.

[121] N. Juristo and A. M. Moreno. *Basics of software engineering experimentation*. Kluwer Academic Publishers, 2001.

[122] S. K. Kachigan. *Statistical analysis – An interdisciplinary introduction to univariate and multivariate methods*. Radius Press, 1982.

[123] 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.

[124] K. Kaminsky and G. D. Boetticher. Defect prediction using machine learners in an implicitly data starved domain. In *Proceedings of the 8th world multi-conference on systemics, cybernetics and informatics*, Orlando, FL, 2004.

[125] S. H. Kan. *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[126] 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.

[127] N. Karunanithi and Y. K. Malaiya. *Neural networks for software reliability engineering, in Handbook of software reliability and system reliability*. McGraw-Hill Inc., Hightstown, NJ, USA, 1996.

[128] 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.

[129] 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.

[130] N. Karunanithi, D. Whitley, and Y. K. Malaiya. Using neural networks in reliability prediction. *IEEE Software*, 9(4):53–59, 1992.

[131] H. G. Kayacik, M. Heywood, and A. N. Zincir-Heywood. On evolving buffer overflow attacks using genetic programming. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO'06)*, pages 1667–1674, New York, NY, USA, 2006. ACM.

[132] H. G. Kayacik, A. N. Z.-Heywood, and M. Heywood. Evolving successful stack overflow attacks for vulnerability testing. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC'05)*, Washington, DC, USA, 2005. IEEE Computer Society.

[133] H. G. Kayacik, A. N. Z.-Heywood, and M. Heywood. Automatically evading IDS using GP authored attacks. In *Proceedings of the IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA'07)*, pages 153–160, New York, NY, USA, 2007. IEE Computer Society.

[134] T. 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.

[135] T. 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.

[136] T. Khoshgoftaar and R. Szabo. Using neural networks to predict software faults during testing. *IEEE Transactions on Reliability*, 45(3):456–462, 1996.

[137] T. M. Khoshgoftaar and E. B. Allen. Logistic regression modeling of software quality. *International Journal of Reliability, Quality and Safety Engineering*, 6(4):303–317, 1999.

[138] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. I. 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.

[139] 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.

[140] 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.

[141] 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.

[142] T. M. Khoshgoftaar and J. C. Munson. Predicting software development errors using software complexity metrics. *IEEE Journal on Selected Areas in Communications*, 8(2):253–261, 1990.

[143] T. M. Khoshgoftaar, J. C. 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.

[144] 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.

[145] 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.

[146] 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.

[147] T. M. Khoshgoftaar and Naeem Seliya. Fault prediction modeling for software quality estimation: comparing commonly used techniques. *Empirical Software Engineering*, 8(3):255–283, 2004.

[148] N. R. Kiran and V. Ravi. Software reliability prediction by soft computing techniques. *Journal of Systems and Software*, 81(4), 2008.

[149] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd. What accuracy statistics really measure? *IEE Proceedings Software*, 148(3), Jun 2001.

[150] B. A. Kitchenham. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-001, UK, July 2007.

[151] 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.

[152] 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.

[153] B. A. Kitchenham, L. M. Pickard, and S. J. Linkman. An evaluation of some design metrics. *Software Engineering Journal*, 5(1):50–58, 1990.

[154] J. Koljonen, M. Mannila, and M. Wanne. Testing the performance of a 2D nearest point algorithm with genetic algorithm generated gaussian distributions. *Expert Systems with Applications*, 32(3):879–889, 2007.

[155] 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. *IEEE International Conference on E-Commerce Technology*, 1, 2002.

[156] M. Kotanchek, G. Smits, and A. Kordon. *Industrial strength genetic programming*. Kluwer, 2003.

[157] J. R. Koza. *Genetic programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.

[158] J. R. Koza and R. Poli. *Search methodologies – Introductory tutorials in optimization and decision support techniques, edited by Edmund K. Burke and Graham Kendall*, chapter 5 – Genetic programming. Springer Science and Business Media, Inc., 233 Spring Street, New York, USA, 2005.

[159] W. Langdon, S. Gustafson, and J. Koza. The genetic programming bibliography, 2009. Last checked 13 Feb 2009.

[160] 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.

[161] M. Lefley and M. J. Shepperd. Using genetic programming to improve software effort estimation based on general data sets. In *Proceedings of the 2003 Conference on Genetic and Evolutionary Computation (GECCO'03)*. ACM, 2003.

[162] Y. Lei and K.-C. Tai. In-Parameter-Order: A test generation strategy for pairwise testing. In *Proceedings of The 3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE'98)*, Washington, DC, USA, 1998. IEEE Computer Society.

[163] 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.

[164] 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.

[165] M. Lipow. Number of faults per line of code. *IEEE Transactions on Software Engineering*, 8(4):437–439, 1982.

[166] Y. Liu and T. 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.

[167] Y. Liu, T. 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*, 2006.

[168] 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.

[169] 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.

[170] S. Luke and L. Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, 2006.

[171] M. R. Lyu. *Handbook of software reliability engineering*. IEEE Computer Society Press and McGraw-Hill, 1996.

[172] 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.

[173] M. R. Lyu and A. Nikora. Applying reliability models more effectively. *IEEE Software*, 9(4), 1992.

[174] C. J. Colbourn M. B. Cohen and A. C. H. Ling. Constructing strength three covering arrays with augmented annealing. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*. IEEE Computer Society, 2003.

[175] 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.

[176] 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.

[177] 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.

[178] T. Mantere and J. T. Alander. Evolutionary software engineering, A review. *Applied Soft Computing*, 5:315–331, 2005.

[179] The MathWorks, Inc. `http://www.mathworks.com`. Last checked 20 Apr 2008.

[180] 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.

[181] 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.

[182] P. McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2):105–156, 2004.

[183] 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.

[184] Z. Michalewicz and D. B. Fogel. *How to solve it: Modern heuristics*. Springer Verlag, second edition, 2004.

[185] F. Mueller and J. Wegener. A comparison of static analysis and evolutionary testing for the verification of timing constraints. *Real-Time Systems*, 21(3):241–268, 1998.

[186] J. Munson and T. M. Khoshgoftaar. Regression modelling of software quality: Empirical investigation. *Journal of Electronic Materials*, 19(6):106–114, 1990.

[187] J. C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, 18(5):423–433, 1992.

[188] J. D. Musa. *Software reliability engineering: More reliable software faster and cheaper*. AuthorHouse, 2nd edition, 2004.

[189] G. J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979.

[190] 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.

[191] 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.

[192] I. J. Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1), 2003.

[193] D. E. Neuman. An enhanced neural network technique for software risk analysis. *IEEE Transactions on Software Engineering*, 28(9):904–912, 2002.

[194] A. P. Nikora. CASRE homepage, 2009. Available at `http://www.openchannelfoundation.org/projects/CASRE\_3.0/`, Last checked 20 Apr 2008.

[195] 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.

[196] K. J. Nurmela. Upper bounds for covering arrays by tabu search. *Discrete Applied Mathematics*, 138(1-2):143–152, 2004.

[197] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, 22(12):886–894, 1996.

[198] 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.

[199] N. Ohlsson, M. Zhao, and M. Helander. Application of multivariate analysis for software fault prediction. *Software Quality Journal*, 7(1):51–66, 1998.

[200] E. Oliveira, A. Pozo, and S. R. 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.

[201] T. J. 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.

[202] T. J. 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.

[203] M. OSullivan, S. Vössner, and J. Wegener. Testing temporal correctness of real-time systems. In *Proceedings of the 6th International Conference on Software Testing Analysis and Review (EuroSTAR'98)*, 1998.

[204] M. Petticrew and H. Roberts. *Systematic reviews in the social sciences: A practical guide*. Wiley-Blackwell, Victoria, Australia, 2005.

[205] L. Pickard, B. Kitchenham, and S. 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.

[206] H. Pohlheim, M. Conrad, and A. Griep. Evolutionary safety testing of embedded control software by automatically generating compact test data sequences. In *SAE 2005 World Congress & Exhibition*, April 2005.

[207] H. Pohlheim and J. Wegener. Testing the temporal behavior of real-time software modules using extended evolutionary algorithms. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference (GECCO'99)*, 1999.

[208] 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).

[209] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza. Genetic programming: An introductory tutorial and a survey of techniques and applications. Technical report CES-475, ISSN: 1744-8050, 2007.

[210] R. S. Pressman. *Software Engineering – A Practitioner's Approach*. McGraw-Hill Higher Education, fifth edition, 2001.

[211] P. Puschner and R. Nossal. Testing the results of static worst-case execution-time analysis. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'98)*, Washington, DC, USA, 1998. IEEE Computer Society.

[212] 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.

[213] S. R. Rakitin. *Software verification and validation for practitioners and managers*. Artech House., Inc., 685 Canton Street, Norwood, MA, USA, second edition, 2001.

[214] 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.

[215] 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.

[216] E. N. Regolin, G. A. de Souza, A. R. T. Pozo, and S. R. Vergilio. Exploring machine learning techniques for software size estimation. In *Proceedings of the International Conference of the Chilean Computer Science Society*, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[217] 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.

[218] C. Robson. *Real world research.* Blackwell Publishing, United Kingdom, second edition, 2002.

[219] S. Russell and P. Norvig. *Artificial intelligence—A modern approach.* Prentice Hall Series in Artificial Intelligence, USA, 2003.

[220] A. C. Schultz, J. J. Grefenstette, and K. A. D. Jong. Adaptive testing of controllers for autonomous vehicles. In *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology (AUV'92)*, 1992.

[221] A. C. Schultz, J. J. Grefenstette, and K. A. De Jong. *Learning to break things: Adaptive testing of intelligent controllers*, chapter G3.5. IOP Publishing Ltd. and Oxford Press, 1997.

[222] 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*, Piscataway, NJ, USA, 2002.

[223] M. Shepperd, M. Cartwright, and G. Kadoda. On building prediction systems for software engineers. *Empirical Software Engineering*, 5(3):175–182, 2000.

[224] M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, 2001.

[225] T. Shiba, T. Tsuchiya, and T. Kikuno. Using artificial life techniques to generate test cases for combinatorial testing. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, Washington, DC, USA, 2004. IEEE Computer Society.

[226] K. K. Shukla. Neuro-genetic prediction of software development effort. *Information and Software Technology*, 42(10):701–713, 2000.

[227] S. Silva. GPLAB—A genetic programming toolbox for MATLAB. `http://gplab.sourceforge.net`. Last checked 30 Mar 2009.

[228] R. Sitte. Comparison of software-reliability-growth predictions: Neural networks vs parametric-recalibration. *IEEE Transactions on Reliability*, 48(3):285–291, 1999.

[229] S. F. Smith. *A learning system based on genetic adaptive algorithms*. PhD thesis, Pittsburgh, PA, USA, 1980.

[230] 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.

[231] 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.

[232] 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.

[233] J. Stardom. Metaheuristics and the search for covering and packing arrays. Master's thesis, Simon Fraser University, B.C., Canada.

[234] C. Stringfellow and A. Amschler Andrews. An empirical method for selecting software reliability growth models. *Empirical Software Engineering*, 7(4):319–343, 2002.

[235] 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.

[236] 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.

[237] J. Tian. An integrated approach to test tracking and analysis. *Journal of Systems and Software*, 35(2):127–140, 1996.

[238] J. Tian. Quality-evaluation models and measurements. *IEEE Software*, 21(3):84–91, 2004.

[239] 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.

[240] 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.

[241] 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.

[242] L. Tian and A. Noore. Computational intelligence methods in software reliability prediction. *Computational Intelligence in Reliability Engineering*, 39:375–398, 2007.

[243] M. Tlili, S. Wappler, and H. Sthamer. Improving evolutionary real-time testing. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*. ACM Press, 2006.

[244] C. Torgerson. *Systematic reviews*. Continuum International Publishing Group, 2003.

[245] N. Tracey, J. Clark, and K. Mander. The way forward for unifying dynamic test case generation: The optimisation-based approach. In *Internation Workshop on Dependable Computing and Its Applications (DCIA'98)*, 1998.

[246] N. Tracey, J. Clark, J. McDermid, and K. Mander. Integrating safety analysis with automatic test data generation for software safety verification. In *Proceedings of 17th International System Safety Conference*. System Safety Society, 1999.

[247] 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.

[248] Y.-W. Tung and W.S. Aldiwan. Automating test case generation for the new generation mission software system. In *Proceedings of the 2000 IEEE Aerospace Conference Proceedings*, 2000.

[249] 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.

[250] A. Veevers and A. C. Marshall. A relationship between software coverage metrics and reliability. *Software Testing, Verification and Reliability*, 4(1):3–8, 1994.

[251] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. Time aware test suite prioritization. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis (ISSTA'06)*, New York, NY, USA, 2006. ACM.

[252] J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, and B. Jones. Systematic testing of real-time systems. In *Proceedings of the 4th International Conference on Software Testing Analysis and Review (EuroSTAR'96)*, 1996.

[253] J. Wegener and M. Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*, 15(3):275–298, 1998.

[254] J. Wegener, P. Pitschinetz, and H. Sthamer. Automated testing of real-time tasks. In *Proceedings of the 1st International Workshop on Automated Program Analysis, Testing and Verification (WAPATV'00)*, Limerick, Ireland, 2000.

[255] J. Wegener, H. Sthamer, B. F. Jones, and D. E. Eyres. Testing real-time systems using genetic algorithms. *Software Quality Control*, 6(2):127–135, 1997.

[256] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[257] I. H. Witten and E. Frank. *Data mining—Practical machine learning tools and techniques*. Morgan Kaufmann Publishers, USA, 2005.

[258] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: An introduction*. Kluwer Academic Publishers, USA, 2000.

[259] A. Wood. Predicting software reliability. *Computer*, 29(11), 1996.

[260] 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.

[261] 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.

[262] X. Yao and Y. Liu. *Search methodologies – Introductory tutorials in optimization and decision support techniques, edited by Edmund K. Burke and Graham Kendall*. Springer Science and Business Media, Inc., 233 Spring Street, New York, USA, 2005. Chapter 12 – Machine learning.

[263] 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.

[264] Y. Zhan and J. A. Clark. Search based automatic test-data generation at an architectural level. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO'04)*. ACM, 2004.

[265] D. Zhang and J. J. P. Tsai. Machine learning and software engineering. *Software Quality Control*, 11(2):87–119, 2003.

[266] 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.

[267] Y. Zhang and J. Yin. Software reliability model by AGP. In *Proceedings of the IEEE International Conference on Industrial Technology*, Piscataway, NJ, United States, 2008.

# Appendix A

# Study quality assessment: Chapter 4

Table A.1: Study quality assessment.

| Criteria |
|---|
| *A*: Are the aims of the research/research questions clearly stated?<br>*B*: Do the study measures allow the research questions to be answered?<br>*C*: Is the sample representative of the population to which the results will generalize?<br>*D*: Is there a comparison group?<br>*E*: Is there an adequate description of the data collection methods?<br>*F*: Is there a description of the method used to analyze data?<br>*G*: Was statistical hypothesis testing undertaken?<br>*H*: Are all study questions answered?<br>*I*: Are the findings clearly stated and relate to the aims of research?<br>*J*: Are the parameter settings for the algorithms given?<br>*K*: Is there a description of the training and testing sets used for the model construction methods? |

(a) Study quality assessment criteria

| | [217] | [78] | [145] | [168] | [140] | [141] | [166] | [215] | [167] |
|---|---|---|---|---|---|---|---|---|---|
| *A* | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *B* | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *C* | ~× | ~√ | ~√ | ~√ | ~√ | ~√ | ~√ | ~√ | ~√ |
| *D* | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *E* | ~√ | ~√ | ~√ | ~√ | ~√ | ~√ | ~√ | ~√ | ~√ |
| *F* | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *G* | × | × | × | × | × | × | × | × | × |
| *H* | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *I* | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *J* | ~√ | √ | ~√ | √ | √ | √ | ~√ | × | √ |
| *K* | √ | √ | √ | √ | √ | √ | √ | √ | √ |

(b) Study quality assessment for software quality classification studies

| | [73] | [71] | [216] | [72] | [45] | [222] | [161] |
|---|---|---|---|---|---|---|---|
| *A* | √ | √ | √ | √ | √ | √ | √ |
| *B* | √ | √ | √ | √ | √ | √ | √ |
| *C* | √ | ~√ | √ | √ | √ | √ | √ |
| *D* | √ | √ | √ | √ | √ | √ | √ |
| *E* | ~√ | √ | √ | √ | √ | √ | √ |
| *F* | √ | √ | √ | √ | √ | √ | √ |
| *G* | × | × | × | × | ~√ | × | × |
| *H* | √ | √ | √ | √ | √ | √ | √ |
| *I* | √ | √ | √ | √ | √ | √ | √ |
| *J* | √ | √ | √ | √ | √ | √ | √ |
| *K* | √ | √ | √ | √ | √ | √ | √ |

(c) Study quality assessment for software CES estimation

| | [123] | [124] | [247] | [266] | [267] | [6] | [59] | [200] |
|---|---|---|---|---|---|---|---|---|
| *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

# Study quality assessment: Chapter 5

Table B.1: Study quality assessment.

| Criteria |
|---|
| *A*: Is the reader able to understand the aims of the research? |
| *B*: Is the context of study clearly stated? |
| *C*: Was there a comparison or control group? |
| *D*: Are the measures used in the study fully defined [150]? |
| *E*: Is there an adequate description of the data collection methods? |
| *F*: Does the data collection methods relate to the aims of the research? |
| *G*: Is there a description of the method used to analyze data? |
| *H*: Are the findings clearly stated and relate to the aims of research? |
| *I*: Is the research useful for software industry and research community? |
| *J*: Do the conclusions relate to the aim and purpose of research defined? |

(a) Study quality assessment criteria

| | [252] | [14] | [255] | [253] | [203] | [245] | [185] | [211] | [207] | [254] | [94] | [91] | [92] | [36] | [243] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *A* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *B* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *C* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | × | × | √ | × | √ |
| *D* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *E* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *F* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *G* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *H* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *I* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| *J* | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |

(b) Study quality assessment for execution time studies

| | [51] | [69] |
|---|---|---|
| *A* | √ | √ |
| *B* | √ | √ |
| *C* | √ | √ |
| *D* | √ | √ |
| *E* | √ | √ |
| *F* | √ | √ |
| *G* | √ | √ |
| *H* | √ | √ |
| *I* | √ | √ |
| *J* | √ | √ |

(c) Study quality assessment for QoS studies

| | [75] | [132] | [44] | [96] | [95] | [131] | [133] |
|---|---|---|---|---|---|---|---|
| *A* | √ | √ | √ | √ | √ | √ | √ |
| *B* | √ | √ | √ | √ | √ | √ | √ |
| *C* | √ | √ | √ | √ | √ | √ | √ |
| *D* | √ | √ | √ | √ | √ | √ | √ |
| *E* | √ | √ | √ | √ | √ | √ | √ |
| *F* | √ | √ | √ | √ | √ | √ | √ |
| *G* | √ | √ | √ | √ | √ | √ | √ |
| *H* | √ | √ | √ | √ | √ | √ | √ |
| *I* | √ | √ | √ | √ | √ | √ | √ |
| *J* | √ | √ | √ | √ | √ | √ | √ |

(d) Study quality assessment for security studies

| | [233] | [55] | [56] | [196] | [174] | [42] | [225] |
|---|---|---|---|---|---|---|---|
| *A* | √ | √ | √ | √ | √ | √ | √ |
| *B* | √ | √ | √ | √ | √ | √ | √ |
| *C* | √ | √ | √ | √ | √ | √ | √ |
| *D* | √ | √ | √ | √ | √ | √ | √ |
| *E* | √ | √ | √ | √ | √ | √ | √ |
| *F* | √ | √ | √ | √ | √ | √ | √ |
| *G* | √ | √ | √ | √ | √ | √ | √ |
| *H* | √ | √ | √ | √ | √ | √ | √ |
| *I* | √ | √ | √ | √ | √ | √ | √ |
| *J* | √ | √ | √ | √ | √ | √ | √ |

(e) Study quality assessment for usability studies

| | [246] | [2] | [24] | [206] |
|---|---|---|---|---|
| *A* | √ | √ | √ | √ |
| *B* | √ | √ | √ | √ |
| *C* | √ | √ | × | × |
| *D* | √ | √ | √ | √ |
| *E* | √ | √ | √ | √ |
| *F* | √ | √ | √ | √ |
| *G* | √ | √ | √ | √ |
| *H* | √ | √ | √ | √ |
| *I* | √ | √ | √ | √ |
| *J* | √ | √ | √ | √ |

(f) Study quality assessment for safety studies

# Appendix C

# Search strings: Chapter 5

- **Abstracts:** (evolutionary OR heuristic OR search-based OR metaheuristic OR optimization OR hill-climbing OR simulated annealing OR tabu search OR genetic algorithms OR genetic programming) AND ("software testing" OR "testing software" OR "test data generation" OR "automated testing" OR "automatic testing") AND (non-functional OR safety OR robustness OR stress OR security OR usability OR integrity OR efficiency OR reliability OR maintainability OR testability OR flexibility OR reusability OR portability OR interoperability OR performance OR availability OR scalability)

- **Titles:** (evolutionary OR heuristic OR search-based OR metaheuristic OR optimization OR hill-climbing OR simulated annealing OR tabu search OR genetic algorithms OR genetic programming) AND (testing) AND (non-functional OR safety OR robustness OR stress OR security OR usability OR integrity OR efficiency OR reliability OR maintainability OR testability OR flexibility OR reusability OR portability OR interoperability OR performance OR availability OR scalability)

- **Keywords:** (evolutionary OR heuristic OR search-based OR metaheuristic OR optimization OR hill-climbing OR simulated annealing OR tabu search OR genetic algorithms OR genetic programming) AND ("software testing" OR "testing software" OR "test data generation" OR "automated testing" OR "automatic testing") AND (non-functional OR safety OR robustness OR stress OR security OR usability OR integrity OR efficiency OR reliability OR maintainability OR testability OR flexibility OR reusability OR portability OR interoperability OR performance OR availability OR scalability)

## ABSTRACT

Software verification and validation activities are essential for software quality but also constitute a large part of software development costs. Therefore efficient and cost-effective software verification and validation activities are both a priority and a necessity considering the pressure to decrease time-to-market and intense competition faced by many, if not all, companies today. It is then perhaps not unexpected that decisions related to software quality, when to stop testing, testing schedule and testing resource allocation needs to be as accurate as possible.

This thesis investigates the application of search-based techniques within two activities of software verification and validation: Software fault prediction and software testing for non-functional system properties. Software fault prediction modeling can provide support for making important decisions as outlined above. In this thesis we empirically evaluate symbolic regression using genetic programming (a search-based technique) as a potential method for software fault predictions. Using data sets from both industrial and open-source software, the strengths and weaknesses of applying symbolic regression in genetic programming are evaluated against competitive techniques. In addition to software fault prediction this thesis also consolidates available research into predictive modeling of other attributes by applying symbolic regression in genetic programming, thus presenting a broader perspective. As an extension to the application of search-based techniques within software verification and validation this thesis further investigates the extent of application of search-based techniques for testing non-functional system properties.

Based on the research findings in this thesis it can be concluded that applying symbolic regression in genetic programming may be a viable technique for software fault prediction. We additionally seek literature evidence where other search-based techniques are applied for testing of non-functional system properties, hence contributing towards the growing application of search-based techniques in diverse activities within software verification and validation.

ISSN 1650-2140
ISBN 978-91-7295-163-1

2009:06