Genetic programming for cross-release fault count predictions in large and complex software projects

Wasif Afzal, Richard Torkar, Robert Feldt, Tony Gorschek

Blekinge Institute of Technology, S-372 25 Ronneby, Sweden

Keywords: Fault prediction, Cross-release, Empirical, Machine learning

Abstract

Software fault prediction can play an important role in ensuring software quality through efficient resource allocation. This could, in turn, reduce the potentially high consequential costs due to faults. Predicting faults might be even more important with the emergence of short-timed and multiple software releases aimed at quick delivery of functionality. Previous research in software fault prediction has indicated that there is a need i) to improve the validity of results by having comparisons among number of data sets from a variety of software, ii) to use appropriate model evaluation measures and iii) to use statistical testing procedures. Moreover, cross-*release* prediction of faults has not yet achieved sufficient attention in the literature. In an attempt to address these concerns, this paper compares the quantitative and qualitative attributes of 7 traditional and machine-learning techniques for modeling the cross-release prediction of fault count data. The comparison is done using extensive data sets gathered from a total of 7 multi-release open-source and industrial software projects. These software projects together have several years of development and are from diverse application areas, ranging from a web browser to a robotic controller software. Our quantitative analysis suggests that genetic programming (GP) tends to have better consistency in terms of goodness of fit and accuracy across majority of data sets. It also has comparatively less model bias. Qualitatively, ease of configuration and complexity are less strong points for GP even though it shows generality and gives transparent models. Artificial neural networks did not perform as well as expected while linear regression gave average predictions in terms of goodness of fit and accuracy. Support vector machine regression and traditional software reliability growth models performed below

average on most of the quantitative evaluation criteria while remained on average for most of the qualitative measures.

1. Introduction

The development of quality software on time and within stipulated cost is a challenging task. One influential factor in software quality is the presence of software faults, which have a potentially considerable impact on timely and cost-effective software development. Thus fault prediction models have attracted considerable interest in research (as shown in Section 2). 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 (Taghi and Naeem, 2002). 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 and also for selective architectural improvements. This paper presents both quantitative and qualitative evaluations of using genetic programming (GP) for cross-release predictions of fault count data gathered from 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 a 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 that is suited to multi-release software development projects.

Regardless of large number of studies on software fault prediction, there is little convergence of results across them. This non-convergence of results is also highlighted by other authors such as (Stefan, Bart, Christophe & Swantje, 2008; Magnus and Per, 2002). This necessitates further research to increase confidence in the results of fault prediction studies. Stefan et al. (2008) identifies three sources of bias in fault prediction studies: use of small and

proprietary data sets, inappropriate accuracy indicators and limited used of statistical testing procedures. Magnus and Per (2002) further highlight the need for the authors to provide information about context, data characteristics, statistical methods and chosen parameters; so as to encourage replications of these studies. These factors, more or less, contribute to a lack of benchmarking in software fault prediction studies. Therefore, *benchmarking*, although recommended by (Susan, Steve & Richard, 2003) as a way to advance research, is still open for further research in software fault prediction studies.

One challenge in having trustworthy software fault prediction models is the nature of typical software engineering data sets. Software engineering data come with certain characteristics that complicate the task of having accurate software fault 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 (Gray and MacDonell, 1997). Therefore, "it is very difficult to make valid assumptions about the form of the functional relationship between the variables" (Lionel, Victor & William, 1992, pp. 931). Moreover, the acceptability of models has seen little success due to lack of meaningful explanation of the relationship among different variables and lack of generalizability of model results (Gray and MacDonell, 1997). Applications of computational and artificial intelligence have attempted to deal with some of these challenges, see e.g. (Zhang and Tsai, 2003), 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 (Yu-Shen and Chin-Yu, 2007).
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. (Tian and Noore, 2005). 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 on a smaller scale than validating existing approaches on a larger scale. In this paper we try to focus on the latter part.

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 (Section 2). Our main focus is on evaluating genetic programming (GP) for cross-release prediction of fault counts on data set 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 industrial software systems (See Section 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 (Goel, 1985). These three models are Goel-Okumoto non-homogeneous Poisson process model (GO) (Goel and Okumoto, 1979), Brooks and Motley's Poisson model (BMP) (Brooks and Motley, 1980) and Yamada's S-Shaped growth model (YAM) (Yamada and Osaki, 1983). 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 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. (Tian and Noore, 2007; Raj and Ravi, 2008, Khoshgoftaar and Yi, 2007) for some examples.

Our main goal is to answer the question:

Is GP a better approach for cross release prediction of fault counts compared to traditional and machine learning approaches 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 4. By applying the model creation approaches described above and by answering the research questions the paper 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. Use of GP for *cross-release* fault predictions.
3. Comparative evaluations with both traditional and machine learning models for *cross-release* prediction of fault count data.

The remainder of this paper is organized as follows. In Section 2, we present the background for this study. Section 3 elaborates on the data collection procedure. Section 4 describes the research questions, while Section 6 provides a brief introduction to the techniques used in the study. Section 5 describes the different evaluation measures used in the study while Section 7 covers the application of different techniques and the corresponding evaluation. Validity evaluation is presented in Section 8 while discussion and conclusions are presented in Section 9.

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" (Khoshgoftaar and Seliya, 2003, pp. 256). The applicable methods include statistical methods (random-time approach, stochastic approach), machine learning methods and mixed algorithms (Venkata, Farokh, Yen and Raymond, 2005). Despite the presence of large number of models, there is little agreement

within the research community about the best model (Stefan et al. 2008). The result is that the prediction problem is seen as being largely unsolvable and NP-hard (Venkata et al., 2005; Martin and Gada, 2001). 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.

Kehan and Khoshgoftaar (2007) 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, Shen & Dunsmore (1988) 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. Taghi, John, Bibhuti, & Gary (1992) 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 (Munson and Khoshgoftaar, 1992), 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 (Lionel, Victor & Christopher, 1993), optimized set reduction classifications (that generates logical expressions representing patterns in the data) were found to be more accurate

than multivariate logistic regression and classification trees in modeling high-risk software components. The less optimistic results of using logistic regression are not in agreement with Khoshgoftaar's study (Khoshgoftaar and Allen, 1999), which supports using logistic regression for software quality classification. Also the study by (Giovanni and Mauro, 2002) used logistic regression to successfully classify faults across homogeneous applications. Victor, Lionel, Walc & Lio (1996) verified that most of Chidamber and Kemerer's object-oriented metrics are useful quality indicators for fault-prone classes. Niclas, Christin & Mary, (1997) investigated the use of metrics for release *n* to identify the most fault prone modules in release *n+1*. In (Niclas, Ming & Mary, 1998), 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, Taghi, Edward, Wendell and John (1999) showed that the classification-tree models based on several product, process and execution measurements were useful in quality classification for successive software releases. Lionel, Walcelio and Wust (2002) 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 (Jeremy and Art, 2007), 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.

We also 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 (Laura, 1991, pp. 212).

These characteristics are also common to data collected from a typical software development process. Karunanithi et al. published several studies (Karunanithi, Malaiya & Whitley, 1991; Nachimuthu, Darrell & Yashwant, 1992; Nachimuthu, Darrell & Yashwant, 1992; Karunanithi and Malaiya, 1996; Karunanithi, 1993) using neural network architectures for software reliability growth modeling. Other examples of studies reporting encouraging results include (Tian and Noore, 2005; Raj and Ravi, 2008; Khoshgoftaar, Pandya, & More, 1992; Khoshgoftaar, Allen, Hudepohl & Aud, 1997; Khoshgoftaar and Szabo, 1996; Tadashi, Yasuhiko & Shunji, 1999; Sitte, 1999; Aljahdali, Sheta & Rine, 2001; Adnan, Yaakob, Anas & Tamjis, 2000; Guo and Lyu, 2004; Nidhi and Pratap, 2005; Ho, Xie & Goh, 2003; Tian and Noore, 2004; Tian and Noore, 2005; Tian and Noore, 2005). Kai-Yuan, Lin, Wei-Dong, Zhou-Yi, & David (2001) 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 and MacDonell, 1997) 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. (Khoshgoftaar and Seliya, 2003) 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.

Tibor, Rudolf & Istvan (2005) used OO metrics for predicting 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 Stefan et al. (2008) also concluded that with respect to classification, there were no significant differences among 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 (Cai, Wen & Zhang, 1991; Cai, Wen & Zhang, 1993; So, Cha & Kwon, 2002; Utkin, Gurov & Shubinsky, 2002), and support vector machines, as in (Tian and Noore, 2007), 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 (Eduardo, Silvia, Aurora & Gustavo, 2005; Eduardo, Aurora & Silvia, 2006; Costa, de Souza, Pozo, & Vergilio, 2007; Yongqiang and Huashan, 2006; Afzal, Torkar & Feldt, 2008; Wasif & Richard, 2008; Wasif and Richard, 2008). The comparisons with traditional software reliability growth models indicate that genetic programming has an edge with respect to predictive accuracy and does not need assumptions common in these traditional models. There are also several studies where genetic programming

has been successfully used for software quality classification (Khoshgoftaar and Yi, 2007; Taghi, Yi & Naeem, 2004).

There are also studies that use a combination of techniques, e.g. (Tian and Noore, 2007), where genetic algorithms are used to determine an optimal neural network architecture and in (Donald, 2002), where principal component analysis is used to enhance the performance of neural networks.

As mentioned in Section 1, very few studies have looked at cross-release predictions of fault data on a large scale. Thomas and Elaine (2002) 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 (Elaine, Thomas & Robert, 2005) 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. 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 involved together represent several man years of development and span a multitude of different software applications targeting e.g. home users, small business users and industrial, embedded systems.

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

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

The industrial fault count data sets come from three large companies specializing in different domains. The first industrial data set (IND01) is from a European company in the space industry. The multi-release software is for an on-board computer used in a satellite system. It consists of about 70,000 lines of manually written C code for drivers and other low-level functions and about 230,000 lines of C code generated automatically from Simulink models. The total number of man-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 tests. 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

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

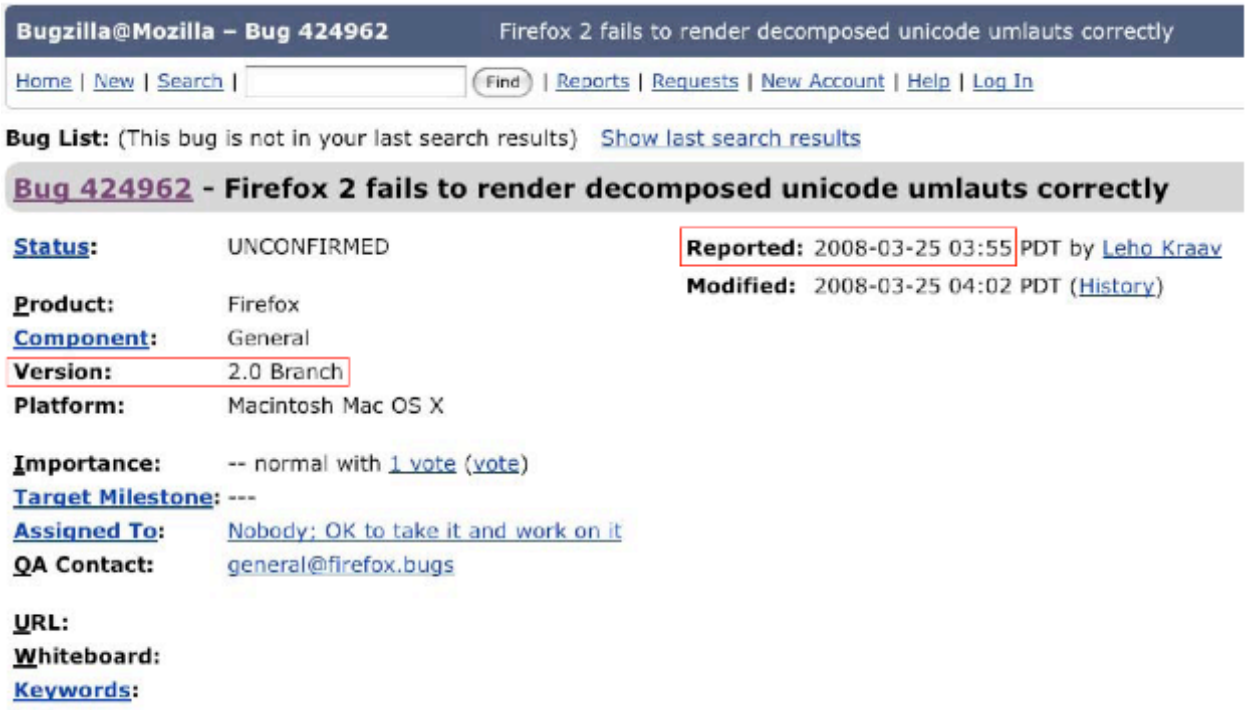
a state-of-the-art self-optimizing motion technology, security and error handling mechanism and advanced user-authorization system. IND03 consists of fault count data from a robotic packaging software. This software comes with an advanced vision technique and integrated conveyor tracking capability; while being open to communicate with any external sensor. The total number of man-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 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 350000 lines of hand written Delphi/Pascal code (90%) and C code (10%). Total development time is about 12000 man-days, 30% of this has been on system tests.

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.

Figure 1. A sample bug report.



As an example, Figure 1 shows a bug report for Mozilla Firefox. For OSStom and OSSmoz, we recorded the data from the `Reported' and `Version' fields as shown in the Figure 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 (Paul, Mary, Jim, Bonnie, & Santhanam, 2004). 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. We were assisted by our industrial partners in provision of the fault count data sets IND01, IND02, IND03 and IND04.

Table 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 Test on 6.0.14	24	20
OSSbsd	4.0, 4.1 (Jan.–Jul. 2007), 4.2 (Oct.–Dec. 2007)	Train on 4.0, 4.1 Test on 4.2	28	12
OSSmoz	1.0, 1.5 (Jul.–Dec. 2005), 2.0 (Jan.–Jun. 2006)	Train on 1.0, 1.5 Test on 2.0	72	24
IND01	4.3.0, 4.3.1, 4.4.0, 4.4.1, 4.5.0 (Oct. 2006–Feb. 2007), 4.5.1 (Mar.–Apr. 2007)	Train on 4.3.0, 4.3.1, 4.4.0, 4.4.1, 4.5.0 Test on 4.5.1	20	8
IND02	5.07, 5.09 (Feb. 2006–Apr. 2007), 5.10 (Feb.–Dec. 2007)	Train on 5.07, 5.09 Test on 5.10	38	11
IND03	5.09, 5.10 (Sept. 2005–Dec. 2007)	Train on 5.09 Test on 5.10	19	11
IND04	3.0, 3.1 (Jan. 2007–Mar. 2008), 3.2 (Sept.–Dec. 2008)	Train on 3.0, 3.1 Test on 3.2	60	16

Table 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 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.

4. Research questions

Before presenting the empirical study in detail, where we compare different fault count modeling methods, we need to detail the specific research questions to be answered. Informally, we want to evaluate if GP can be a better approach for cross-release prediction of fault count data in general when compared with 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 the GP-evolved model as compared with traditional and machine learning models for cross-release predictions?

RQ 2: What are the levels of *predictive accuracy* of GP-evolved model for cross-release predictions as compared with traditional and machine learning models?

RQ 3: What is the *prediction bias* of the GP-evolved model for cross-release prediction of fault count data as compared with traditional and machine learning models?

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

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. (Shepperd, Cartwright & Kadoda, 2000). Since a comparison of different measures is out of scope for this paper, 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 (Nikora and Lyu, 1995).

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

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 (Stringfellow and Andrews, 2002; Matsumoto, Inoue, Kikuno & Torii, 1988). 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 (Taghi, Naeem & Nandini, 2006; Malaiya, Karunanithi & Verma, 1990). We further test a particular model's bias, which gives an indication of whether the model is prone to overestimation or underestimation (Malaiya et al., 1990). To measure a particular model's bias, we examine the distribution of residuals to compare models as suggested in (Kitchenham, Pickard, MacDonell & Shepperd, 2001; Lesley, Barbara & Susan, 1999). We also formally test for significant differences between competing prediction systems as recommended in e.g. (Shepperd et al, 2000). 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,

$$J = \frac{mn}{d} \max_{-\infty<t<+\infty} \{|F_m(t) - G_n(t)|\} \quad (1)$$

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 detailed description of the test, see (Hollander and Wolfe, 1999).

Prequential likelihood ratio (PLR). PLR is used to investigate the relative plausibility of the predictions from two models (Abdel-Ghaly, Chan & Littlewood, 1986). 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}, ... T_{j+n}:

$$PL_n = \prod_{i=j+1}^{j+n} \hat{f}_i(t_i) \tag{2}$$

The PLR of two prediction systems, *A* and *B*, is then the running product of ratio of their successive on-step ahead predictions $\hat{f}^{A}_{j}(t_j)$ and $\hat{f}^{B}_{j}(t_j)$ respectively (Sarah and Bev, 1996):

$$PLR_i^{AB} = \prod_{j=s}^{j=i} \frac{\hat{f}^{A}_{j}(t_j)}{\hat{f}^{B}_{j}(t_j)} \tag{3}$$

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 (Sarah and Bev, 1996).

Absolute average error (AAE). The AAE is given by,

$$AAE = \frac{1}{n} \sum_{i=1}^{n} | y_i - \hat{y}_i | \tag{4}$$

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.

Absolute relative error (ARE). The ARE is given by,

$$ARE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_i|} \quad (5)$$

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. The 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) ≥ 0.75 which means that at least 75% of the estimates are within the range of 25% of the actual values (Jos and Javier, 2000).

Distribution of residuals. To measure a particular model bias, we examine the distribution of residuals to compare models (Kitchenham et al., 2001; Shepperd et al., 2000). It has the convenience of applying significance tests and visualizing differences in absolute residuals of competing models using box plots.

5.2. Qualitative evaluation

In addition to the quantitative evaluation factors, there are other qualitative criteria, which need to be accounted for when assessing the usefulness of a particular modeling technique. Qualitative criterion-based evaluation evaluates each method based on conceptual requirements (Gray and MacDonell, 1997). One or more of these requirements might influence model selection. We use the following qualitative criteria (Gray and MacDonell, 1997; Carolyn et al. 2000; Michael & Allen, 1992; Burgess and Lefley, 2001), 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?

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 (Silva, 2007) (for running GP), Weka software version 3.4.13 (Witten & Frank, 2005) (for running ANN, SVM and LR) and SMERFS3 version 2 (Farr, 2009) (for running GO, YAM and BMP).

6.1. Genetic programming (GP)

GP is an evolutionary computation technique and is an extension of genetic algorithms (Koza, 1992). 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 (Bäck, Fogel & Michalewicz, 2000). 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 (Burke and Kendall, 2005). 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 (Silva, 2007).

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, $\sum_{i=1}^{n} | e_i - e_i^* |$, 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.

Table 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,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

The control parameters that were chosen for the GP system are shown in Table 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 (Silva, 2007). For a new population, the parents and off springs 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 (Silva, 2007).

6.2. Artificial neural networks (ANN)

The development of artificial neural networks is inspired by the interconnections of biological neurons (Russell and Norvig, 2003). These neurons, also called nodes or units, are connected by direct links. These links are associated with numeric weights, which show both the strength and sign of the connection (Russell and Norvig, 2003). 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 (Witten & Frank, 2005). Recurrent neural networks feeds

its output back into its 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 networks. 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 multilayer perceptron implemented in Weka software version 3.4.13 was 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.

6.3. Support vector machine (SVM)

Support vector regression uses 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 (Russell and Norvig, 2003). For regression, the basic idea is to discard the deviations up to a user specified parameter ε (Witten & Frank, 2005). Apart from specifying ε, 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 support vector regression implemented in Weka software version 3.4.13 was used for training which uses Smola and Scholkopf sequential minimization algorithm (Smola and Schölkopf, 2004) for training. More details on support vector regression can be found in (Smola and Schölkopf, 2004; Gunn 1998).

6.4. Linear regression (LR)

The linear regression used in the study performs a standard least-squares linear regression (Kachigan, 1982). 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. The general form of a simple linear regression is,

$$y=\propto+\beta x \qquad (6)$$

where $\propto$ (y-intercept) and β (slope) are unknown and are estimated from data.

6.5. Traditional software reliability growth models

As discussed in Section 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 (Goel and Okumoto, 1979; Brooks and Motley, 1980; Yamada and Osaki, 1983).

The Goel-Okumoto non-homogeneous Poisson process model (GO) (Goel and Okumoto, 1979) is given by,

$$m(t) = a\,[1 - e^{-bt}] \qquad (7)$$

while Yamada's S-shaped growth model (YAM) (Yamada and Osaki, 1983) is also a non-homogeneous Poisson process model given by,

$$m(t) = a\,(1- (1+bt)\, e^{-bt}) \qquad (8)$$

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 (Jeff, 1996):

$$P[X(t) = n] = \frac{[m(t)]^n e^{-m(t)}}{n!} \quad (9)$$

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. Specifically, Brooks and Motley's Poisson model (BMP) (Brooks and Motley, 1980) with 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,

$$P[X = n_i] = \begin{cases} \frac{(N_i \phi_i)^{n_i} e^{-N_i \phi_i}}{} \\ \phi_i = 1 - (1 - \phi)^{t_i} \end{cases} \quad (10)$$

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

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 in to training and test sets. The training set is used to build the models while the *independent* (hold-out) test set is used to evaluate the models' performance. The training and test data sets are given in Table 1. For all our data sets, we had the complete data set available upfront because they were based on projects having a historical pool of data. Therefore we were required to select a split mechanism that was impartial and also preserve the time-series nature of data. A generally accepted practice for data mining algorithms is to hold out one-third of the data for testing and use the remaining two-thirds for training (Witten & Frank, 2005), however, owing to the *cross-release* nature of this study, we resorted to a training set comprising of fault data from previous releases of the software under consideration while the testing set was comprised of fault data from the forthcoming release.

The performance is assessed both quantitatively (goodness of fit, predictive accuracy, model bias) and qualitatively (ease of configuration, solution transparency, generality and complexity). The independent variable in our case is the week number while the corresponding dependent variable is the count of faults. Week number is taken as the independent variable because it is controllable and potentially has an effect on the dependent variable, i.e. the count of faults, in which the effect of the treatment is measured. All our data sets were uni-variate, which suited the cross-release nature of this study since collecting multivariate data across releases is quite difficult and also would have severely complicated the cross-release modeling. The design type of our experiment is one factor with more than two treatments (Box, Hunter & Hunter, 1978). 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.

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 represent the same population. The null hypothesis here is that the predicted and the actual fault count data have the same probability distribution (Hollander and Wolfe, 1999) i.e.,

$$H_0 : [F(t) = G(t), \text{ for every } t] \qquad (11)$$

At significance level $\propto = 0.05$, if the K-S statistic J is greater than or equal to the critical value $J_{\propto}$, the null hypothesis (Eq. 11) is rejected in favor of the alternate hypothesis, i.e. that the two samples do *not* have the same probability distribution.

Table 3. Results of applying Kolmogorov-Smirnov test. The bold values indicate $J < J_{\propto}$, (--) indicates lack of model convergence, $J_{\propto}$ is the critical J value at $\propto$=0.05.

Table 3: Results of applying Kolmogorov-Smirnov test. The bold values indicate $J<J_{\alpha}$, (–) indicates lack of model convergence, J_{α} is the critical J value at α=0.05

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

Table 3 shows the results of applying K-S test statistic for each technique for every data set. The (--) in the Table 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 $J_{\propto}$ are shown in bold in Table 3. It is evident from Table 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 $\propto$ = 0.05, GP is better in terms of having statistically significant goodness of fit on more data sets than other, competing, techniques.

Table 4. Summary statistics for K-S test showing the mean, median, min and max corresponding to the respective number of data sets.

Technique	No. of data sets	K-S test statistic			
		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

Table 4 shows the summary of K-S test statistic for all the techniques. Since some techniques did not converge for some data sets, the number of data sets applicable for techniques is different. GP, ANN, LR and BMP were able to converge for all seven data sets. However, the same did not happen with other techniques, as can be seen from the second column of Table 4. We can observe that in comparison with ANN, LR and BMP with seven data sets each, GP appears to be a better technique (shows a comparatively closer fit to the set of observations) when ranked based on mean and median.

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, both in terms of K-S test statistic and ranking based on mean and median, on more data sets.

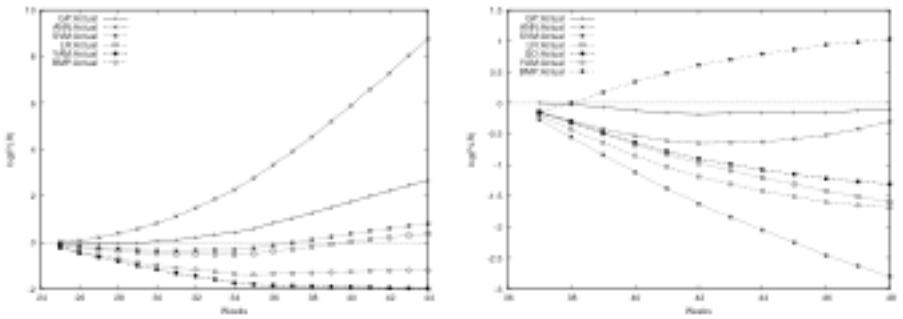
7.2. Evaluation of predictive accuracy

Table 5. *log*(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

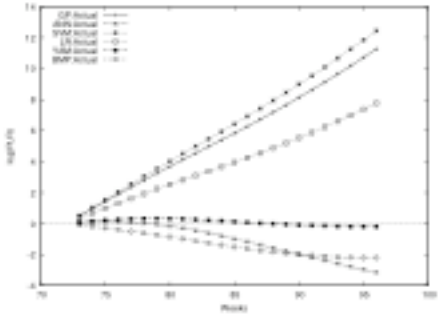
Table 5 shows the final *log* result of the running product of the ratio of the successive one-step ahead predictions of actual fault count data and other techniques' prediction. Since the actual time distribution of weekly/monthly fault count data is chosen as the reference, the PLR values closer to 0 are better. We observe from Table 5 that the *log*(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 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.

Figure 2. *log*(PLR) plots for the data sets OSStom, OSSbsd, OSSmoz, IND01, IND02, IND03 and IND04.

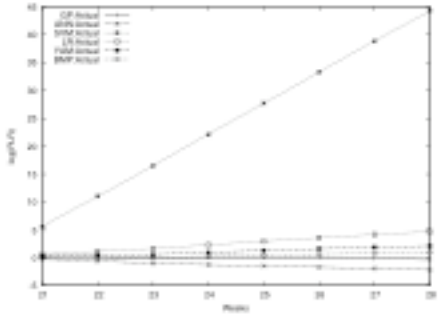


(a) OSStom.

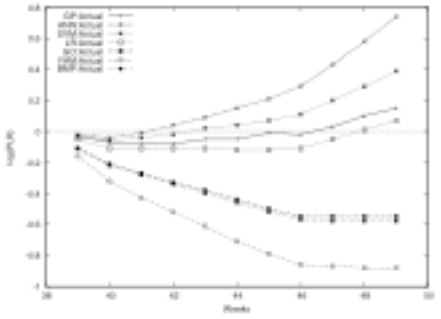
(b) OSSbsd.



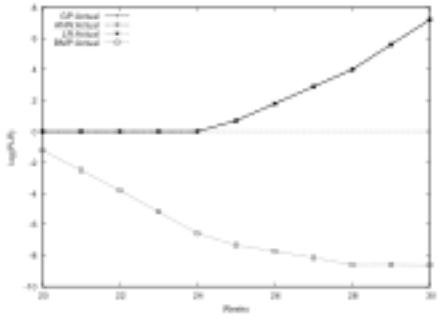
(c) OSSmoz.



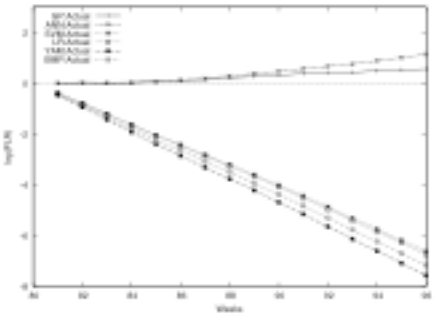
(d) IND01.



(e) IND02.



(f) IND03.



(g) IND04.

Figure 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 2). We see that for OSStom (Figure 2a), the prediction curves for LR and SVM are closer to the reference in

comparison with other curves. For OSSbsd (Figure 2b), the prediction curve for GP follows the reference more closely than other curves. Same behavior is also evident for IND01, IND03 and IND04 (Figures 2d, 2e and 2g). However, for OSSmoz (Figure 2c), YAM is better at following the reference compared to any other curve, while for IND03 (Figure 2f), the curves for GP, ANN and LR are much closer to the *log*(PLR) of actual fault count data. Overall, GP was able to show more consistent predictive accuracy, across four of the seven data sets.

Table 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

Table 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.

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 for following pairs of AAE samples: GP vs. ANN, GP vs. SVM, GP vs. LR, GP vs. YAM and GP vs. BMP. The corresponding *p*-values for these tests came out to be 0.27, 0.02, 0.13, 0.03 and 0.22 respectively. At significance level of 0.05, the result indicated that the null hypothesis is rejected for GP vs. SVM and GP vs. YAM, while

showing that there is no statistically significant difference between the AAE means of GP, ANN, LR and BMP at the 0.05 significance level.

Table 7. Summary statistics for AAE showing the mean, median, min and max corresponding to the respective number of data sets.

Technique	No. of data sets	AAE statistic			
		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

Apart from statistical testing, Table 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 7 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 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

Table 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, GP vs. YAM and GP vs. BMP. The corresponding *p*-values for these tests came out to be 0.18, 0.07, 0.15, 0.29 and 0.17 respectively. This shows that, for significance level of 0.05, there is no statistical difference between the ARE means of GP, ANN, SVM, LR, YAM and BMP.

Table 9. Summary statistics for ARE showing the mean, median, min and max corresponding to the respective number of data sets.

Technique	No. of data sets	ARE statistic			
		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

Apart from statistical testing, we can observe from Table 9 that having a ranking based on both mean and median; GP has the lowest value in comparison with ANN, LR and BMP having 7 data sets each.

Table 10. Pred(0.25) calculation for different techniques for all data sets. (--) shows lack of model convergence.

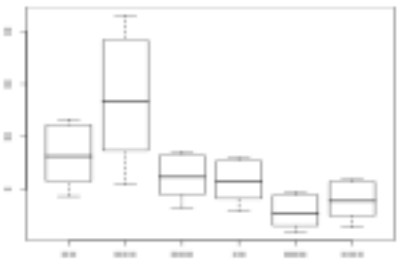
	Sample size	GP (%)	ANN (%)	SVM (%)	LR (%)	GO (%)	YAM (%)	BMP (%)
OSStom	20	100	30	100	100	–	99	100
OSSbsd	12	100	100	50	100	100	100	100
OSSmoz	24	41.67	100	0	16.67	–	100	100
IND01	8	100	37.5	100	0	–	37.5	100
IND02	11	100	45.45	100	100	100	100	100
IND03	11	45.45	45.45	–	44.45	–	–	18.18
IND04	16	100	100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

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

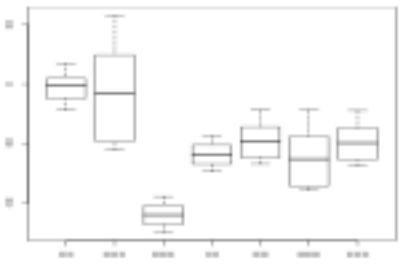
We conclude that while the statistical tests for AAE and ARE do not give us a clear indication of a particular technique being (statistically) significantly better compared to other techniques, the summary statistics (Tables 7 and 9) together with the evaluation of pred(0.25) and PLR show that the predictive accuracy of GP for cross-release prediction of fault count data is promising in comparison with other techniques.

7.3. Evaluation of model bias

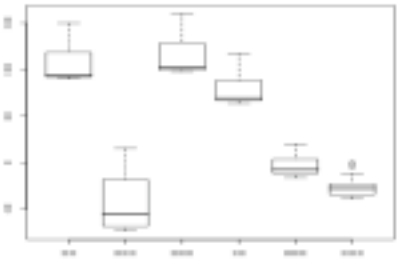
Figure 3. Charts showing Box plots of residuals for the seven data sets.



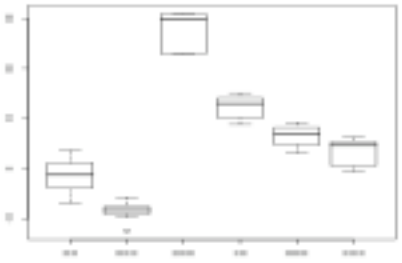
(a) Residuals for OSStom.



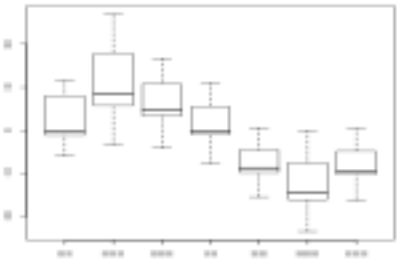
(b) Residuals for OSSbsd.



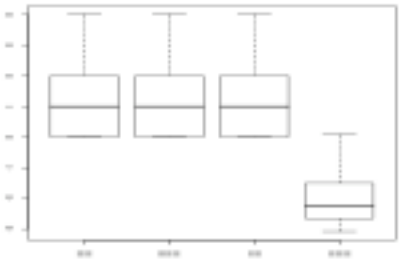
(c) Residuals for OSSmoz.



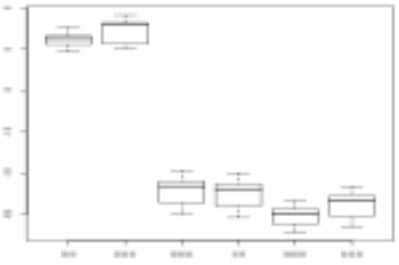
(d) Residuals for IND01.



(e) Residuals for IND02.



(f) Residuals for IND03.



(g) Residuals for IND04.

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. For OSSbsd (Figure 3b) and IND04 (Figure 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 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 3a), SVM and LR are better placed than rest of the techniques while for OSSmoz (Figure 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 3e) where GP and LR show a good trade-off between length and actual position of the box plot.

Table 11. Kruskal-Wallis statistic *h* for different data sets for testing difference in residuals. ν is the degrees of freedom.

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

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 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.

Table 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

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 12. The table shows that except for four cases, the p-values were found to be less than 0.05, which rejects 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 12.)

We conclude that in terms of model bias, the examination of residuals show the greater consistency of GP as compared with other traditional and machine learning models in having predictions that result in smaller box plots that are positioned near the 0-mark. Further,

application of the Wilcoxon rank sum test shows that except for four combinations (GP:ANN-OSSbsd, GP:SVM-IND02, GP:LR-IND02, GP:ANN-IND02), there is sufficient evidence to show that the residuals from GP are different from those of other competing techniques.

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 (Michael & Allen, 1992). 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) (Farr, 2009; Nikora, 2009), 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" (Gray and MacDonell, 1997, pp. 435). 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)) (12)

where *x* is the independent variable and *minus*, *times*, *sin*, *cos*, *log* represents the function set (as outlined in Table 2). While eq. 12 is transparent, it is still difficult to explain the relationship between the independent and dependent variables. Therefore, simplification of resulting GP solutions is important which has to do with finding solutions with less nodes to make the results understandable analytically.

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. 12), a phenomenon known as bloating. Although there are different ways to control this (see e.g. Sean and Liviu, 2006), 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 that 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 (Gray and MacDonell, 1997). The machine learning and evolutionary models certainly scores high in this respect in comparison with traditional methods.

8. Validity evaluation

There can be different threats to the validity of the empirical results (Claes et al., 2000).

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 (Juristo and Moreno, 2001); however, facing some criticism lately (Ioannidis, 2005). 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 6.1) is similar to the quantitative evaluation measures for comparing different techniques (Subsection 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 releases. We were not sure if this is an influential factor in our study. We plan to investigate this in the future. A similar threat is the data set size available for training and subsequently testing the models. This is our limitation that the actual fault counts of different releases determined the data set size that varied across different software.

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. Also, 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. One reason for this is that one of the industrial data sets was only available at this level of detail. While some studies (e.g. Wood, 1996) 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 for data that is more regular, with less frequent changes. However, it is hard to predict such effects and without further study we cannot determine if it is really a threat.

Construct validity, is concerned with the relationship between the 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 commercial software projects that 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 representative one, being based on techniques which have different modeling mechanism and are currently active fields of research.

9. Discussion and conclusions

In this paper, we compared the cross-release predictions of fault count data from models evolved using GP with common machine learning and traditional models. 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 (Carolyn et al. 2000) and should be taken into account along with the quantitative performance.

The results of K-S test statistic showed statistically significant goodness of fit for the GP-evolved models for the majority of the data sets. The predictive accuracy of the competing

models was assessed using PLR, AAE, ARE and pred *(l)*. Using PLR, GP-evolved models showed consistently better predictive accuracy across four of the seven data sets. For the measure of AAE and ARE, GP models were able to give the respective lowest values for largest number of data sets. Finally, GP models were able to meet the standard criterion of pred (0.25) ≥ 75 for stable model predictions for the most number of data sets (compared to other techniques). This observation regarding the accuracy of GP predictions is in agreement with Arthur (2006) where the GP models performed with acceptable quality within 20% outside the training data range. We also assessed the model bias of competing techniques using distribution of residuals, and in that analysis GP models showed a tendency of having less biased predictions as compared with other traditional and machine learning approaches. 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 (Dolado, Fernandez, Otero & Urkola, 1999). This means that neural networks require frequent retraining 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.
4. Potentially poor generalizability outside the range of the training data (Kordon, Smits, Jordaan & Rightor, 2002).

In contrast, GP possesses certain unique characteristics considering the above issues. Symbolic regression using GP is flexible because of its ability to adjust 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 (Kotanchek, Smits & Kordon, 2003).

This might also be useful for an easy integration in existing industrial work processes whereby only those variables could be used.

Table 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							

A brief summary of the relative performance of different techniques is presented in Table 13. The performance indicators are given as to summarize the detailed evaluation done in the study based on several measures (Section 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 7.4. Table 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

[5] The parsimonious factor takes into account that the model with the smallest number of parameters is usually the best.

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 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; they however lack 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. It 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 shows the flexibility of GP, i.e. suits 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 paper followed a standard\slash 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 to compare the relative short-term and long-term predictive strength of the GP-evolved models for different lengths of training data. Lastly, evaluation of ensemble methods presents another opportunity for future research.

References

Abdel-Ghaly, A. A., Chan, P. Y., & Littlewood, B. (1986). Evaluation of competing software reliability predictions. *IEEE Trans. Softw. Eng., 12*(9), 950-967.

Adnan, W. A., Yaakob, M., Anas, R., & Tamjis, M. R. (2000). *Artificial neural network for software reliability assessment.* Paper presented at the Proceedings of TENCON 2000.

Afzal, W., Torkar, R., & Feldt, R. (2008). *Prediction of fault count data using genetic programming.* Paper presented at the proceedings of INMIC 2008 IEEE International Multitopic Conference.

Aljahdali, S. H., Sheta, A., & Rine, D. (2001). *Prediction of software reliability: A comparison between regression and neural network non-parametric models.* Paper presented at the proceedings of ACS/IEEE International Conference on Computer Systems and Applications, 2001.

Arthur, K. (2006). Evolutionary computation at Dow Chemical. *SIGEVOlution, 1*(3), 4-9.

Arthur Karl, K., Guido, F. S., & Mark, E. K. (2007). *Industrial evolutionary computing*. Paper presented at the Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation.

Bäck, T., Fogel, D. B., & Michalewicz, Z. (2000). *Evolutionary computation 1–Basic algorithms and operators*: Taylor & Francis Group.

Barry, B., & Victor, R. B. (2001). Software Defect Reduction Top 10 List. *Computer, 34*(1), 135-137.

Box, G. E. P., Hunter, W. G., & Hunter, J. S. (1978). *Statistics for experimenters: An introduction to design, data analysis, and model building*: Wiley-Interscience.

Brooks, W., & Motley, R. (1980). *Analysis of discrete software reliability models.* IBM Federal Systems.

Bugzilla homepage. (Last checked: October 2008). from http://www.bugzilla.org

Burgess, C. J., & Lefley, M. (2001). Can genetic programming improve software effort estimation? A comparative evaluation. *Information and Software Technology, 43*(14), 863-873.

Burke, E. K., & Kendall, G. (2005). *Search methodologies–Introductory tutorials in optimization and decision support techniques*: Springer Science and Business Media.

Cai, K.-Y. (1996). *Introduction to fuzzy reliability, chapter 8 of Fuzzy methods in software reliability modeling*: Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers.

Cai, K.-Y., Wen, C., & Zhang, M. (1991). A critical review on software reliability modeling. *Reliability engineering and system safety, 32*(3), 357-371.

Cai, K.-Y., Wen, C., & Zhang, M. (1993). A novel approach to software reliability modeling. *Microelectronics and Reliability, 33*(15), 2265-2267.

Carolyn, M., Gada, K., Martin, L., Keith, P., Chris, S., Martin, S., et al. (2000). An investigation of machine learning based prediction systems. *J. Syst. Softw., 53*(1), 23-29.

Claes, W., Per, R., Martin, H, Magnus, C. O., Björn, R., Wesslén, A. (2000). *Experimentation in software engineering: An introduction*: Kluwer Academic Publishers.

Costa, E. O., de Souza, G. A., Pozo, A. T. R., & Vergilio, S. R. (2007). Exploring Genetic Programming and Boosting Techniques to Model Software Reliability. *Reliability, IEEE Transactions on, 56*(3), 422-434.

Dolado, J. J., Fernandez, L., Otero, M. C., & Urkola, L. (1999). *Software effort estimation: The elusive goal in project management*. Paper presented at the Proceedings of the International Conference on Enterprise Information Systems.

Donald, E. N. (2002). An Enhanced Neural Network Technique for Software Risk Analysis. *IEEE Trans. Softw. Eng., 28*(9), 904-912.

Eduardo, O., Aurora, P., & Silvia Regina, V. (2006). *Using Boosting Techniques to Improve Software Reliability Models Based on Genetic Programming*. Paper presented at the Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence.

Eduardo Oliveira, C., Silvia, R. V., Aurora, P., & Gustavo, S. (2005). *Modeling Software Reliability Growth with Genetic Programming*. Paper presented at the Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering.

Farr, W. (Last checked March 2009). SMERFS3 homepage. from http://www.slingcode.com/smerfs/downloads/

Giovanni, D., & Mauro, P. (2002). *An empirical evaluation of fault-proneness models*. Paper presented at the Proceedings of the 24th International Conference on Software Engineering.

Goel, A. L. (1985). Software Reliability Models: Assumptions, Limitations, and Applicability. *Software Engineering, IEEE Transactions on, SE-11*(12), 1411-1423.

Goel, A. L., & Okumoto, K. (1979). Time dependent error detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability, R-28*(3), 206-211.

Gray, A. R., & MacDonell, S. G. (1997). A comparison of techniques for developing predictive models of software metrics. *Information and Software Technology, 39*(6), 425-437.

Gunn, S. R. (1998). *Support vector machines for classification and regression.* School of electronics and computer science, University of Southampton.

Guo, P., & Lyu, M. R. (2004). A pseudoinverse learning algorithm for feedforward neural networks with stacked generalization applications to software reliability growth data. *Neurocomputing, 56*, 101-121.

Ho, S. L., Xie, M., & Goh, T. N. (2003). A study of the connectionist models for software reliability prediction. *Computers & Mathematics with Applications, 46*(7), 1037-1045.

Hollander, M., & Wolfe, D. A. (1999). *Non-parametric statistical methods*: John Wiley and Sons, Inc.

Ioannidis, J. P. A. (2005). Why most published research findings are false. *PLoS Medicine, 2*(8), 696-701.

Jeff, T. (1996). An integrated approach to test tracking and analysis. *J. Syst. Softw., 35*(2), 127-140.

Jeremy, G., & Art, F. (2007). Data mining static code attributes to learn defect predictors. *IEEE Trans. Softw. Eng., 33*(1), 2-13.

Jos, & Javier, D. (2000). A validation of the component-based method for software size estimation. *IEEE Trans. Softw. Eng., 26*(10), 1006-1021.

Juristo, N., & Moreno, A. M. (2001). *Basics of software engineering experimentation*: Kluwer Academic Publishers.

Kachigan, S. K. (1982). *Statistical analysis - An interdisciplinary introduction to univariate and multivariate methods*: Radius Press.

Kai-Yuan, C., Lin, C., Wei-Dong, W., Zhou-Yi, Y., & David, Z. (2001). On the neural network approach in software reliability modeling. *J. Syst. Softw., 58*(1), 47-62.

Karunanithi, N. (1993). *A neural network approach for software reliability growth modeling in the presence of code churn.* Paper presented at the Proceedings of Fourth International Symposium on Software Reliability Engineering, 1993.

Karunanithi, N., & Malaiya, Y. K. (1996). *Neural networks for software reliability engineering , in*

Handbook of software reliability and system reliability: McGraw-Hill Inc., Hightstown, NJ, USA.

Karunanithi, N., Malaiya, Y. K., & Whitley, D. (1991). *Prediction of software reliability using neural networks*. Paper presented at the proceedings of the 1991 International Symposium on Software Reliability Engineering.

Kehan, G., & Khoshgoftaar, T. M. (2007). A Comprehensive Empirical Study of Count Models for Software Fault Prediction. *IEEE Transactions on Reliability, 56*(2), 223-236.

Khoshgoftaar, T., Allen, E., Hudepohl, J., & Aud, S. (1997). Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks, 8*(4).

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

Khoshgoftaar, T. M., Pandya, A. S., & More, H. B. (1992). *A neural network approach for predicting software development faults.* Paper presented at the Proceedings of the Third International Symposium on Software Reliability Engineering, 1992.

Khoshgoftaar, T. M., & Seliya, N. (2003). Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques. *Empirical Software Engineering, 8*(3), 255-283.

Khoshgoftaar, T. M., & Szabo, R. M. (1996). Using neural networks to predict software faults during testing. *IEEE Transactions on Reliability, 45*(3), 456-462.

Khoshgoftaar, T. M., & Yi, L. (2007). A Multi-Objective Software Quality Classification Model Using Genetic Programming. *IEEE Transactions on Reliability, 56*(2), 237-245.

Kitchenham, B. A., Pickard, L. M., MacDonell, S. G., & Shepperd, M. J. (2001). What accuracy statistics really measure? *Software, IEE Proceedings -, 148*(3), 81-85.

Kordon, A., Smits, G., Jordaan, E., & Rightor, E. (2002). *Robust soft sensors based on integration of genetic programming, analytical neural networks, and support vector machines.* Paper presented at the Proceedings of the CEC '02 Proceedings of the 2002 Congress on Evolutionary Computation, 2002.

Kotanchek, M., Smits, G., & Kordon, A. (2003). Industrial strength genetic programming. In *Genetic programming theory and practise* (pp. 239-256): Kluwer.

Koza, J. (1992). *Genetic programming: on the programming of computers by means of natural selection*: MIT Press.

Langdon, W. B. (2008). *A Field Guide to Genetic Programing*: Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk.

Laura Ignizio, B. (1991). Introduction to artificial neural systems for pattern recognition. *Comput. Oper. Res., 18*(2), 211-220.

Lesley, P., Barbara, K., & Susan, L. (1999). *An Investigation of Analysis Techniques for Software Datasets*. Paper presented at the Proceedings of the 6th International Symposium on Software Metrics.

Leung, H., & Varadan, V. (2002). *System modeling and design using genetic programming.* Paper presented at the Proceedings of First IEEE International Conference on Cognitive Informatics, 2002.

Li, P. L., Shaw, M., & Herbsleb, J. (2003). *Selecting a defect prediction model for maintenance resource planning and software insurance.* Paper presented at the proceedings of the Fifth Workshop on Economics-Driven Software Research.

Lionel, C. B., Victor, R. B., & Christopher, J. H. (1993). Developing Interpretable Models with Optimized set Reduction for Identifying High-Risk Software Components. *IEEE Trans. Softw.*

Eng., 19(11), 1028-1044.

Lionel, C. B., Victor, R. B., & William, M. T. (1992). A Pattern Recognition Approach for Software Engineering Data Analysis. *IEEE Trans. Softw. Eng., 18*(11), 931-942.

Lionel, C. B., Walcelio, L. M., Wust, J. (2002). Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Trans. Softw. Eng., 28*(7), 706-720.

Magnus, C. O., & Per, R. (2002). *Experience from Replicating Empirical Studies on Prediction Models*. Paper presented at the Proceedings of the 8th International Symposium on Software Metrics.

Malaiya, Y. K., Karunanithi, N., & Verma, P. (1990). *Predictability measures for software reliability models.* Paper presented at the Proceedings of COMPSAC 90 Fourteenth Annual International Computer Software and Applications Conference, 1990.

Martin, S., & Gada, K. (2001). Comparing Software Prediction Techniques Using Simulation. *IEEE Trans. Softw. Eng., 27*(11), 1014-1022.

Matsumoto, K., Inoue, K., Kikuno, T., & Torii, K. (1988). *Experimental evaluation of software reliability growth models.* Paper presented at FTCS-18, Digest of Papers, the proceedings of Eighteenth International Symposium on Fault-Tolerant Computing, 1988.

Michael, R. L., & Allen, N. (1992). Applying Reliability Models More Effectively. *IEEE Softw., 9*(4), 43-52.

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

Nachimuthu, K., Darrell, W., & Yashwant, K. M. (1992). Using Neural Networks in Reliability Prediction. *IEEE Softw., 9*(4), 53-59.

Nachimuthu, K., Darrell, W., & Yashwant, K. M. (1992). Prediction of Software Reliability Using Connectionist Models. *IEEE Trans. Softw. Eng., 18*(7), 563-574.

Niclas, O., Ann Christin, E., & Mary, H. (1997). Early Risk-Management by Identification of Fault-prone Modules. *Empirical Softw. Engg., 2*(2), 166-173.

Niclas, O., & Hans, A. (1996). Predicting Fault-Prone Software Modules in Telephone Switches. *IEEE Trans. Softw. Eng., 22*(12), 886-894.

Niclas, O., Ming, Z., & Mary, H. (1998). Application of multivariate analysis for software fault prediction. *Software Quality Control, 7*(1), 51-66.

Nidhi, G., & Manu Pratap, S. (2005). Estimation of software reliability with execution time model using the pattern mapping technique of artificial neural network. *Comput. Oper. Res., 32*(1), 187-199.

Nikora, A. P. (2009). CASRE homepage. from http://www.openchannelfoundation.org/projects/CASRE_3.0

Nikora, A. P., & Lyu, M. R. (1995). *An experiment in determining software reliability model applicability.* Paper presented at the Proceedings of the Sixth International Symposium on Software Reliability Engineering, 1995.

Palu, S. (Last checked November 2008). Instance-based learning: A Java implementation. from http://www.developer.com/java/other/article.php/10936_1491651_1

Paul Luo, L., Mary, S., Jim, H., Bonnie, R., & Santhanam, P. (2004). *Empirical evaluation of defect projection models for widely-deployed production software systems*. Paper presented at the Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering.

Raj Kiran, N., & Ravi, V. (2008). Software reliability prediction by soft computing techniques. *Journal of Systems and Software, 81*(4), 576-583.

Raymond, E. S. (1999). *Cathedral and the bazaar*: O'Reily & Associates.

Reformat, M., Pedrycz, W., & Pizzi, N. J. (2003). Software quality analysis with the use of

computational intelligence. *Information and Software Technology, 45*(7), 405-417.

Elaine, J. W., Thomas J. Ostrand & Robert, M. B. (2005). Predicting the Location and Number of Faults in Large Software Systems. *IEEE Trans. Softw. Eng., 31*(4), 340-355.

Russell, S., & Norvig, P. (2003). *Artificial intelligence-A modern approach*: Prentice Hall Series in Artificial Intelligence.

Sarah, B., & Bev, L. (1996). Techniques for prediction analysis and recalibration. In *Handbook of software reliability engineering* (pp. 119-166): McGraw-Hill, Inc.

Sean, L., & Liviu, P. (2006). A comparison of bloat control methods for genetic programming. *Evol. Comput., 14*(3), 309-344.

Shepperd, M., Cartwright, M., & Kadoda, G. (2000). On Building Prediction Systems for Software Engineers. *Empirical Software Engineering, 5*(3), 175-182.

Silva, S. (2007). GPLAB - A Genetic Programming Toolbox for MATLAB. from http://gplab.sourceforge.net (Last checked 27 February 2009)

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

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing, 14*(3), 199-222.

So, S. S., Cha, S. D., & Kwon, Y. R. (2002). Empirical evaluation of a fuzzy logic-based software quality prediction model. *Fuzzy Sets and Systems, 127*(2), 199-208.

Stefan, L., Bart, B., Christophe, M., & Swantje, P. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Trans. Softw. Eng., 34*(4), 485-496.

Stringfellow, C., & Andrews, A. A. (2002). An Empirical Method for Selecting Software Reliability

Growth Models. *Empirical Software Engineering, 7*(4), 319-343.

Susan Elliott, S., Steve, E., & Richard, C. H. (2003). *Using benchmarking to advance research: a challenge to software engineering*. Paper presented at the Proceedings of the 25th International Conference on Software Engineering.

Tadashi, D., Yasuhiko, N., & Shunji, O. (1999). Optimal software release scheduling based on artificial neural networks. *Ann. Softw. Eng., 8*(1-4), 167-185.

Taghi, M. K., Edward, B. A., Wendell, D. J., & John, P. H. (1999). *Classification Tree Models of Software Quality Over Multiple Releases*. Paper presented at the Proceedings of the 10th International Symposium on Software Reliability Engineering.

Taghi, M. K., John, C. M., Bibhuti, B. B., & Gary, D. R. (1992). Predictive Modeling Techniques of Software Quality from Software Measures. *IEEE Trans. Softw. Eng., 18*(11), 979-987.

Taghi, M. K., & Naeem, S. (2002). *Tree-Based Software Quality Estimation Models For Fault Prediction*. Paper presented at the Proceedings of the 8th International Symposium on Software Metrics.

Taghi, M. K., Naeem, S., & Nandini, S. (2006). An empirical study of predicting software faults with case-based reasoning. *Software Quality Control, 14*(2), 85-111.

Taghi, M. K., Yi, L., & Naeem, S. (2004). *Module-Order Modeling using an Evolutionary Multi-Objective Optimization Approach*. Paper presented at the Proceedings of the 10th International Symposium on Software Metrics.

Thomas, J. O., & Elaine, J. W. (2002). *The distribution of faults in a large industrial software system*. Paper presented at the Proceedings of the 2002 ACM SIGSOFT international symposium on software testing and analysis.

Tian, L., & Noore, A. (2004). Software reliability prediction using recurrent neural network with

Bayesian regularization. *International Journal of Neural Systems, 14*(3), 165-174.

Tian, L., & Noore, A. (2005). Dynamic software reliability prediction: An approach based on Support Vector Machines. *International Journal of Reliability, Quality and Safety Engineering, 12*(4), 309-321.

Tian, L., & Noore, A. (2005). Evolutionary neural network modeling for software cumulative failure time prediction. *International Journal of Reliability, Quality and Safety Engineering, 87*(1), 45-51.

Tian, L., & Noore, A. (2005). On-line prediction of software reliability using an evolutionary connectionist model. *Journal of Systems and Software, 77*(22), 173-180.

Tian, L., & Noore, A. (2007). Computational intelligence methods in software reliability prediction. *Computational Intelligence in Reliability Engineering, 39*, 375-398.

Tibor, G., Rudolf, F., & Istvan, S. (2005). Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Trans. Softw. Eng., 31*(10), 897-910.

Utkin, L., Gurov, S., & Shubinsky, M. (2002). A fuzzy software reliability model with multiple-error introduction and removal. *International Journal of Reliability, Quality and Safety Engineering, 9*(3).

Venkata, U. B. C., Farokh, B. B., Yen, I. L., & Raymond, A. P. (2005). *Empirical Assessment of Machine Learning based Software Defect Prediction Techniques*. Paper presented at the Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems.

Victor, R. B., Lionel, C. B., Walc, & lio, L. M. (1996). A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Trans. Softw. Eng., 22*(10), 751-761.

Wasif, A., & Richard, T. (2008). *A Comparative Evaluation of Using Genetic Programming for*

Predicting Fault Count Data. Paper presented at the Proceedings of The Third International Conference on Software Engineering Advances.

Wasif, A., & Richard, T. (2008). *Suitability of Genetic Programming for Software Reliability Growth Modeling*. Paper presented at the Proceedings of the International Symposium on Computer Science and its Applications.

Witten, I. H., & Frank, E. (2005). *Data mining-Practical machine learning tools and techniques*: Morgan Kaufmann Publishers.

Wood, A. (1996). Predicting software reliability. *Computers, 29*(11), 69-77.

Yamada, S., Ohba, M., & Osaki, S. (1983). S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability, R-32*(5), 475-478.

Yongqiang, Z., & Huashan, C. (2006). *Predicting for MTBF Failure Data Series of Software Reliability by Genetic Programming Algorithm*. Paper presented at the Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications - Volume 01.

Yu, T. J., Shen, V. Y., & Dunsmore, H. E. (1988). An Analysis of Several Software Defect Models. *IEEE Trans. Softw. Eng., 14*(9), 1261-1270.

Yu-Shen, S., & Chin-Yu, H. (2007). Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. *J. Syst. Softw., 80*(4), 606-615.

Zhang, D., & Tsai, J. (2003). Machine Learning and Software Engineering. *Software Quality Journal, 11*(2), 87-119.