# Search-based prediction of fault-slip-through in large software projects

Wasif Afzal, Richard Torkar and Robert Feldt
*Blekinge Institute of Technology, Ronneby, Sweden*
*Email: waf|rto|rfd@bth.se*

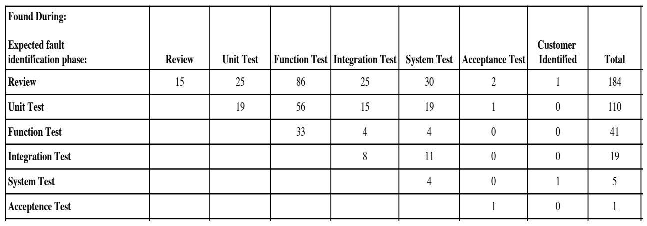
Greger Wikstrand
*KnowIT YAHM Sweden AB*
*Email: greger.wikstrand@yahm.se*

***Abstract*—A large percentage of the cost of rework can be avoided by finding more faults earlier in a software testing process. Therefore, determination of which software testing phases to focus improvements work on, has considerable industrial interest. This paper evaluates the use of five different techniques, namely particle swarm optimization based artificial neural networks (PSO-ANN), artificial immune recognition systems (AIRS), gene expression programming (GEP), genetic programming (GP) and multiple regression (MR), for predicting the number of faults slipping through unit, function, integration and system testing phases. The objective is to quantify improvement potential in different testing phases by striving towards finding the right faults in the right phase. We have conducted an empirical study of two large projects from a telecommunication company developing mobile platforms and wireless semiconductors. The results are compared using simple residuals, goodness of fit and absolute relative error measures. They indicate that the four search-based techniques (PSO-ANN, AIRS, GEP, GP) perform better than multiple regression for predicting the fault-slip-through for each of the four testing phases. At the unit and function testing phases, AIRS and PSO-ANN performed better while GP performed better at integration and system testing phases. The study concludes that a variety of search-based techniques are applicable for predicting the improvement potential in different testing phases with GP showing more consistent performance across two of the four test phases.**

## I. Introduction

Software quality is most commonly referred to as conformance to both functional and non-functional requirements. Presence of software faults is usually taken to be an important factor in software quality, a factor that shows an absence of quality. There have been different types of software quality evaluation models proposed in software engineering literature, all of them with the objective of accurately quantifying software quality (see [1] for a classification of these models).

We know that faults are cheaper to find and remove earlier in the software development process [2]. Software testing is the major fault-finding activity therefore much research has focused on making the software testing process as efficient as possible. One way to improve testing process efficiency is to avoid unnecessary rework by finding more faults earlier. The Faults-Slip-Through (FST) concept [3], [4] is one way of providing quantified decision support to reduce the effort spent on rework. The Faults-Slip-Through (FST) concept is used for determining whether or not a fault slipped through the phase where it should have been found. The term phase refers to any phase in a typical software development life cycle. However the most interesting and industry-supported applications of FST measurement are during testing. This is because a defined testing strategy within any organization implicitly classifies faults whereby certain types of faults might be targeted by certain strategies. FST is essentially a fault classification approach and focuses on when it is cost-effective to find each fault. Depending on the FST numbers for each testing phase (e.g. unit, function, integration and system), improvement potentials can be determined by calculating the difference between the cost of faults in relation to what the fault cost would have been if none of them would have slipped through the phase where they were supposed to be found. Thus, FST is a way to provide quantified decision support to reduce the effort spent on rework. This reduction in effort is due to finding faults earlier in a cost-effective way.



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

Figure 1. An example FST matrix.

One way to visualize FST for different software testing phases is using an FST matrix (Figure 1). The columns in Figure 1 represent the phases in which the faults were found (*Found During*), whereas the rows represent the phases where the faults should have been found (*Expected fault identification phase*). For example 56 of the faults that were found in function testing should have been found during unit testing.

Considering the initial successful results of implementing FST across different organizations within Ericsson [3], [4], it is interesting to investigate how to use FST measurement to provide additional decision support for project management. Staron and Meding [5] highlight that the prediction of FST can be a refinement to their proposed approach for predicting the number of defects in the defect database. Similarly

Damm [3] highlight that FST can potentially be used to support software fault predictions. This additional decision support would be introduced in order to make the software development more *predictable*, i.e. we can better cope with the conflicting demands of doing more with less, doing it faster and doing it with higher quality [6].

Therefore with the goal of providing decision support based on FST, we set out to investigate the following research question:

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

We are particularly interested in evaluating the accuracy of predictions using search-based techniques [7]. The motivation for applying these techniques is that these are non-linear, data-driven and self-adaptive approaches, having the ability to model potentially complex relationships between input and output data. Traditional statistical methods like the autoregressive moving average have difficulties in modeling non-linearities and unknown explicit relationships among variables. Search-based techniques, on the other hand, are independent of satisfying any data assumptions and do not require the definition of the functional form of the relationship upfront. The above properties make these techniques general and flexible for data-driven modeling tasks. We have added multiple regression as another technique to better compare the prediction ability of search-based techniques.

Secondly, we use metrics related to the status of various work packages, testing progress and fault-inflow at the project level as our independent variables. We present the results of evaluating the use of multiple regression (MR) and four search-based techniques (artificial immune recognition system (AIRS), gene expression programming (GEP), genetic programming (GP) and particle swarm optimization based artificial neural network (PSO-ANN)) to predict FST in each of the four testing phases (unit, function, integration, system) of a large-scale project carried out by a company developing mobile platforms and wireless semiconductors. The predictions are compared for differences in models' residuals, goodness of fit and absolute relative error values. The comparative results indicate that while MR is not able to perform as good as the search-based techniques, GP performs better for two testing phases (integration and system) and AIRS, PSO-ANN perform better for unit and function testing phases respectively.

The remainder of the paper is organized as follows. Section II summarizes related work. Section III describes the research context and Section IV discusses the variables, feature selection and evaluation measures used. The search-based techniques used in this paper are discussed in Section V. Results are presented in Section VI while Section VII contains the discussion. Validity evaluation and conclusions make up Sections VIII and IX, respectively.

## II. Related work

There are a number of modeling mechanisms to predict the quality of software. Due to the definition of software quality in many different ways, previous studies have focused on predicting different but related dependent variables of interest; examples include predicting for defect density [8], software defect content estimation [9], fault-proneness [10] and software reliability prediction in terms of time-to-failure [11]. In addition, several independent variables have been used to predict the above dependent variables of interest [1]; examples include prediction using size and complexity metrics [12] and prediction using testing metrics [13]. The actual prediction is performed using a variety of approaches, and can broadly be classified into statistical regression techniques, machine learning approaches and mixed algorithms [14]. Increasingly, evolutionary and bio–inspired approaches are being used for software quality classification [15], [16].

This study is different from the above software quality evaluation studies. First, the dependent variable of interest here is the number of faults slipping through to various testing phases with the aim of triggering corrective actions for avoiding unnecessary rework late in software testing. Second, we make use of several independent variables at the *project* level, i.e. variables depicting work status, testing progress status and fault-inflow. A study by Staron and Meding [5] makes use of project planning and testing status variables for predicting weekly fault inflow. The current study, although taking inspiration from this study, is different in terms of purpose, use of independent and dependent variables and prediction techniques employed.

## III. Research context

The data used in this study comes from two large projects at a telecommunication company that develops mobile platforms and wireless semiconductors. The projects are aimed at developing platforms introducing new radio access technologies written using C. The average number of persons involved in these projects is approximately 250. The data from one of the projects is used as a baseline to train the models while the data from the second project is used to evaluate the models' results. We have data from 45 weeks of the baseline project to train the models while we evaluate the results on data from 15 weeks of an on-going project.

The management of these projects is based on the concepts of tollgates, milestones, steering points and checkpoints to manage and control project deliverables. Tollgates represent long-term business decisions while milestones are predefined events at the operating work level. The monitoring of these milestones is an important element of the project management model. Steering points are defined to coordinate multiple parallel platform projects, e.g. handling priorities between different platform projects. The checkpoints are defined in the development process to define the work status in a process. At the operative work level, software development is structured around work packages (WPs). These work packages are defined during the project planning phase. The work packages are defined to implement change requests or a subset of a use-case, thus the definition of work packages is driven by the functionality to be developed. An essential feature of work packages is that these allow for simultaneous work on different modules of the project at the same time by multiple teams. Since different modules might get affected by developing a single work package, it is difficult to obtain consistent metrics at the module level.

The structure of a project into work packages present an obvious choice for the prediction models since the metrics at work package level have stable values and hold a more direct and intuitive meaning for the company employees. At our subject company the work status of various work packages is grouped using a graphical integration plan (GIP) document. The GIP maps the work packages' status over multiple time-lines that might indicate different phases of software testing or overall project progress. There are different status rankings of the work packages such as number of work packages planned to be delivered for system integration testing.

Table I
VARIABLES OF INTEREST FOR THE PREDICTION MODELS.

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

## IV. RESEARCH METHOD

Our research method is to apply five different techniques to the task of predicting FST in four testing phases based on input data collected from a telecommunication company developing mobile platforms and wireless semiconductors. The input data is made up of different variables divided into 4 sets (Table I), i.e., fault in-flow, status rankings of work packages, faults-slip-through and test case progress.

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

We analyzed the dependencies among variables using scatter plots to identify correlated variables, i.e. variables that potentially measure the same attribute and thus using only one of them would be enough. We were especially interested in visualizing the relationship between the measures of status rankings of work packages, test case progress, fault in-flow and rest of the measures related to status rankings of work packages and test case progress. The pair-wise scatter plots of the above attributes showed a tendency of non-linear relationships. Hence, we calculated the Spearman rank-order correlation coefficient to find if there were any strong indications of correlation among the variables. The Spearman rank-order correlation coefficient values varied between 0.2 and 0.6, indicating that the pairs of variables were not strongly correlated.

We then used the statistical analysis technique of kernel principal component analysis (KPCA) [17] to reduce the number of independent variables to a smaller set that would still capture the original information in terms of explained variance in the data set. The role of original variables in determining the new factors (principal components) is determined by loading factors. Variables with high loadings contribute more in explaining the variance. The results of applying the Gaussian kernel, KPCA (Table II) showed that the first four components explained 97% of the variability in the data set. We did not include the faults-slip-through measures in the KPCA since these are the attributes we are interested in predicting. In each of the four components, all the variables contributed with different loadings, with the exception of two, namely *number of network signaling test cases planned* and *number of network signaling test cases executed*. Hence, we excluded these two variables and used the rest for predicting the fault-slip-through in different testing phases.

The predictive accuracy of different techniques is compared using absolute residuals (i.e. $|\text{actual-predicted}|$) [18],

Table II
THE LOADINGS AND EXPLAINED VARIANCE FROM FOUR PRINCIPAL COMPONENTS.

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

[19] and absolute relative error metric [20]. The goodness of fit of the results from different techniques is assessed using the Kolmogorov-Smirnov (K-S) test. For the K-S test we use $\alpha = 0.05$ and if the K-S statistic $J$ is greater or equal than the critical value $J_\alpha$, we infer that the two samples did not have the same probability distribution and hence do not represent significant goodness of fit.

## V. TECHNIQUES USED

This section discusses the five techniques used in this study for FST prediction.

### A. Particle swarm optimization based artificial neural network (PSO-ANN)

The development of artificial neural networks is inspired by the interconnections of biological neurons [21]. Selecting proper ANN architecture parameters is one of the important decisions for designing the ANN model. Different training schemes for ANN have been proposed, the most used being the back-propagation algorithms [22]. However, back-propagation suffers from certain drawbacks [23] e.g. slow convergence, local minima and lack of robustness. Recently, evolutionary optimization methods have been used for neural network training since the search space of a multilayer feed forward neural network with a non-linear activation function is complex and believed to have many local and global minima [24].

Kennedy and Eberhart introduced PSO in 1995 [25] inspired by the coordinated search for food by a swarm of birds, that can be used for parameter estimation of an ANN. The idea behind PSO is that a swarm of particles move through a multidimensional search space for finding a global optimum (e.g. minimum or maximum of a given objective function). Several improvements of the basic PSO algorithm have been proposed, one of them by Trelea [26]. According to [26], the expression for position and velocity update for particle $i$ of the dimension $d$ is given by:

$$\begin{aligned} v_{id}(t) = a * v_{id}(t-1) + b * (p_{id} - x_{id}(t-1)) + \\ + b * (p_{gd} - x_{id}(t-1)) \\ x_{id}(t) = c * x_{id}(t-1) + d * v_{id}(t) \end{aligned}$$

where $v_{id}(t)$ = velocity of the $i^{th}$ particle of the $d^{th}$ dimension, $p_{id}$ = best position for $i^{th}$ particle of the $d^{th}$ dimension, $x_{id}(t)$ = position of the $i^{th}$ particle of the $d^{th}$ dimension, $p_{gd}$ = global best position of the $d^{th}$ dimension. The parameter $a$ represents the inertia weight that determines the influence of old velocity. The parameter $b$ is the acceleration coefficient that determines the influence of local best position and the global best position. Trelea [26] recommends two sets of parameter values for $a$ and $b$ after simulation experiments, i.e. ($a = 0.6$, $b = 1.7$) and ($a = 0.73$, $b = 1.49$). These two are commonly referred to as TreleaI and TreleaII. The parameters $c$ and $d$ affect a particle's new position and are commonly set to 1.

A three layer feed forward neural network model has been used in this study. The number of layers and the number of nodes at each layer is determined through experimentation. The final ANN structure consisted of one input layer, two hidden layers and one output layer. The two hidden layers consisted of 3 and 5 nodes while the output layer consisted of 1 node. The number of independent variables in the problem determined the number of input nodes. The hyperbolic tangent sigmoid and linear transfer functions have been used for the hidden and output nodes respectively. The PSO variation Trealea II, implemented as part of the MATLAB PSO toolbox [27], has been used in this study with the number of particles in the swarm set to 25 and number of iterations set to 2000. The mean squared error has been used as the fitness function.

### B. Artificial immune recognition system (AIRS)

The concepts of artificial immune systems have been used to produce a supervised learning system called artificial immune recognition system (AIRS) [28].

Based on immune network theory, [29] developed a resource limited artificial immune system and introduced the metaphor of an artificial recognition ball (ARB), a collection of similar B-cells. B-cells are anti-body secreting cells that is a response of the immune system against the attack of a disease. For the resource-limited AIS, a predefined number of resources exists. The ARBs compete based on their stimulation level. Least stimulated cells are removed while an ARB having higher stimulation value could claim more resources. The AIRS is based on the similar idea of a resource-limited system but many of the network principles were abandoned in favor of a simple population-based model [28].

Table III
GEP CONTROL PARAMETERS.

| *Control Parameter* | *Value* |
|---|---|
| Population size | 50 |
| Genes per chromosome | 4 |
| Gene head length | 8 |
| Termination condition | 2000 generations |
| Functions | $\{+,-,*,/,\text{sin,cos,log,sqrt}\}$ |
| Tree initialization | Random |
| Mutation rate, Inversion rate, IS transposition rate, Root transposition rate, Gene transposition rate, One-point recombination rate, Two-point recombination rate, Gene recombination rate | 0.04, 0.1, 0.1, 0.1, 0.1, 0.3, 0.3, 0.1 |
| Selection method | roulette-wheel |

The AIRS algorithm consists of five steps [28]: *1) Initialization*, *2) Memory cell identification and ARB generation*, *3) Competition for resources and development of a candidate memory cell*, *4) Memory cell introduction* and *5) Classification*. The details of each of these steps can be found in [28] and are omitted due to space constraints.

The WEKA plug-in for AIRS [30] has been used with the following parameters: Affinity threshold = 0.2, clonal rate = 10, hypermutation rate = 2, knn = 3, mutation rate = 0.1, stimulation value = 0.9 and total resources = 150.

## *C. Gene expression programming (GEP)*

Gene expression programming (GEP) is an evolutionary algorithm introduced by Ferreira [31]. The individuals making up a population in GEP are encoded as linear strings of fixed length that are later expressed as nonlinear expression trees of different sizes and shapes. Thus GEP brings a separation between genotype (the linear chromosomes) and the phenotype (the expression trees). GEP uses karva notation [31] to encode solutions where each gene has a head composed of functions and terminals, and a tail composed only of terminals [31]. The GEP algorithm begins with a random generation of chromosomes to form an initial population. The chromosomes are expressed as trees and evaluated for fitness (Mean squared error, $\sum_{i=1}^{N}(P_i - T_i)^2$ where $P_i$ and $T_i$ are $i^{th}$ predicted and target values). If the termination criterion is not already reached, the selection process selects best-fit individuals for making a new population for the next generation. Genetic diversity is inculcated in the new population using genetic operators of replication, mutation, inversion, insertion sequence (IS) transposition, root transposition, gene transposition, gene recombination, one-and-two point recombination [31], [32]. The process is iterated for a certain number of generations until the termination criterion is reached. The details of genetic operators are omitted due to space constraints and can be found in [31].

The parameter settings for GEP used in this study are shown in Table III.

Table IV
GP CONTROL PARAMETERS.

| *Control parameter* | *Value* |
|---|---|
| Population size | 50 |
| Termination condition | 2000 generations |
| Function set | $\{+,-,*,/,\text{sin,cos,log,sqrt}\}$ |
| Tree initialization | Ramped half-and-half method |
| Probabilities of crossover, mutation, reproduction | 0.8, 0.1, 0.1 |
| Selection method | roulette-wheel |

## *D. Genetic programming (GP)*

GP, an evolutionary computation technique, is an extension of genetic algorithms [33]. 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 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 [7]. 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. The details of genetic operators are omitted due to space constraints but can be found in [34]. The GP programs were evaluated according to the sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} \mid e_i - e_i^{'} \mid$, where $e_i$ is the actual fault count data, $e_i^{'}$ is the estimated value of the fault count data and $n$ is the size of the data set used to train the GP models. The control parameters that were chosen for the GP system are shown in Table IV.

## *E. Multiple regression (MR)*

Multiple regression is an extension of simple least-square regression for more than one independent (predictor) variables to estimate the values of the dependent (criterion) variable. The general form of the equation is:

$$y' = a + b_1x_1 + b_2x_2 + \ldots + b_kx_k$$

where $y'$ is the predicted value of the dependent variable, $x_1$,$x_2$, …, $x_k$ are independent (predictor) variables and $a$, $b$ are the coefficients that must be determined from sample data. As in simple regression, least square solution is used to determine the best regression equation.

Table VI
ABSOLUTE RELATIVE ERROR, ARE, VALUES FOR DIFFERENT TECHNIQUES AT UNIT, FUNCTION, INTEGRATION AND SYSTEM TEST PHASES.

| | PSO-ANN | AIRS | GEP | GP | MR |
|---|---|---|---|---|---|
| Unit testing | 2.66 | 0.81 | 0.95 | 1.16 | 5.34 |
| Function testing | 2.01 | 4.51 | 2.95 | 3.15 | 4.36 |
| Integration testing | 0.99 | 0.56 | 1.61 | 0.36 | 1.73 |
| System testing | 0.43 | 0.7 | 0.59 | 0.04 | 0.06 |

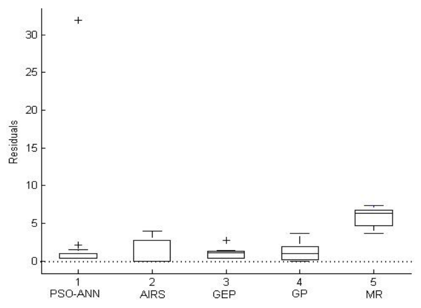
## VI. RESULTS AND ANALYSIS

The box plots of the residuals for faults-slip-through at the *unit testing phase* (from the five techniques: PSO-ANN, AIRS, GEP, GP and MR), are shown in Figure 2(a).

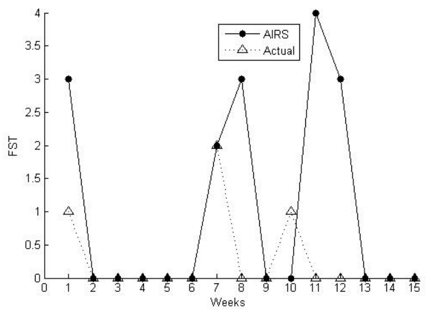
It can be observed that the box plots for PSO-ANN, GEP and GP are narrower than those of AIRS and MR, indicating that the AIRS and MR residuals are more spread out as compared to the other techniques. The box plots for PSO-ANN and GEP are smaller in comparison with that of GP. Since the residual box plots were skewed, we resorted to the non-parametric Kruskal-Wallis test to examine any statistical differences between the residuals and to confirm the trend observed from the box plots (Figure 2(a)). The result of the Kruskal-Wallis test ($p = 0.0000000025$) suggested that it is possible to reject the null hypothesis of all samples being drawn from the same population at significance level, $\alpha = 0.05$. This is to suggest that at least one sample median is significantly different from the others. We used Wilcoxon rank sum test to investigate which of the techniques' residuals differ significantly. The null hypothesis of samples being drawn from identical distributions was rejected at $\alpha = 0.05$ for MR:PSO-ANN, MR:GEP, MR:GP, MR:AIRS. The corresponding p-values for these pairs appear in Table VII. The p-values of rest of the pair-wise comparisons (PSO-ANN:AIRS, PSO-ANN:GEP, PSO-ANN:GP, AIRS:GEP, AIRS:GP, GEP:GP) suggested that there were no significant differences between the model residuals. This suggests that while the group of search-based techniques do not differ significantly pair-wise for residuals, MR residuals are significantly different. We also measured the goodness of fit for predictions from each technique relative to the actual FST at the unit testing phase. Table V shows the K-S statistic $J$ for each of the five techniques. The AIRS technique showed statistically significant goodness of fit which has to do with an exact match of actual FST data on 9 out of 15 instances (Figure 2(b)). Table VI shows the ARE measure for each of the five techniques at the unit test phase. AIRS gives the least ARE value, followed by GEP and GP.

The box plots of the residuals from the five techniques for FST at the *function testing phase* are shown in Figure 2(c). We can observe that there is a greater spread of box plots for each of the techniques as compared with those at the unit testing phase. The plots for each of the techniques are also farther away from the 0 mark on the $y$-axis, with PSO-ANN being the closest of all but having a much larger spread as compared with other box plots. To test for any significant differences between the residuals, the results of applying Kruskal-Wallis test ($p = 0.00022$) suggested that the null hypothesis of all the samples being drawn from the same population can be rejected at $\alpha = 0.05$. This is to suggest that at least one sample median is significantly different from the others. To further investigate which of the techniques' residuals differ, we used the Wilcoxon rank sum test for pair-wise comparisons. Table VII shows the significant p-values for pair-wise comparisons where the null hypothesis of samples being drawn from identical distributions was rejected. Except for the pairs GEP:GP and MR:AIRS, all pair-wise model residuals differ significantly. This confirms that there is much variability in the residuals at the function testing phase for every pair of the techniques except for GEP:GP and MR:AIRS. The pairs GEP:GP and MR:AIRS do not differ significantly but as is evident from their box plots (Figure 2(c)) they both are not in close proximity of mark 0 on the $y$-axis. The goodness of fit of predictions from each of the techniques in relation to the actual FST during function testing is measured using the K-S test. The results appear in Table V. PSO-ANN has statistically significant goodness of fit which is also evident from Figure 2(d) where the PSO-ANN curve is closer to the actual FST curve at the function testing phase. The box plot of PSO-ANN in Figure 2(c) also suggests that the median is much nearer to the 0 mark than other techniques. PSO-ANN is also able to give best ARE values in comparison with other techniques (Table VI).

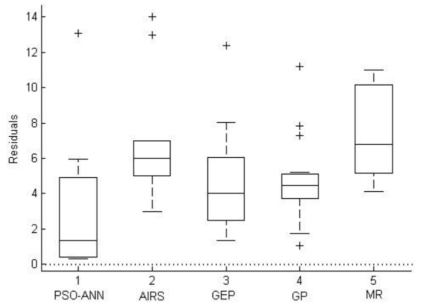
The residual box plots from the five techniques for FST at the *integration testing phase* are shown in Figure 2(e). The box plots in this case are narrow for all the techniques with GP having a median closest to the 0 mark on the $y$-axis while MR box plot appears farthest away. This indicates that while there does not seem to be much difference in absolute residuals of the four search-based techniques, the residuals of MR appear to be differently placed. In order to confirm this, the Kruskal-Wallis test was performed and the result ($p = 0.000077$) showed that at least one sample median is significantly different from others. As with previous two test phases the Wilcoxon rank sum test for pair-wise comparisons was performed. The p-values in Table VII confirmed that for all pairs involving MR (i.e. MR:PSO-ANN, MR:AIRS, MR:GEP, MR:GP) we can reject the null hypothesis of samples being drawn from identical distributions. For all the pair-wise comparisons of residuals of four search-based techniques, there were no significant differences between samples. The results of Kolmogorov-Smirnov test for measuring the goodness of fit of predictions in relation to the actual FST at the integration testing phase are given in Table V. Table V shows that AIRS and GP have statistically significant goodness of fit, with their K-S



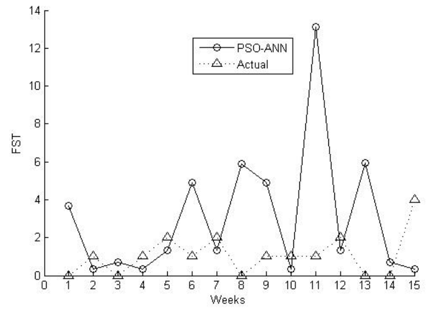
(a) Box plots of the residuals for each technique at the unit testing phase.



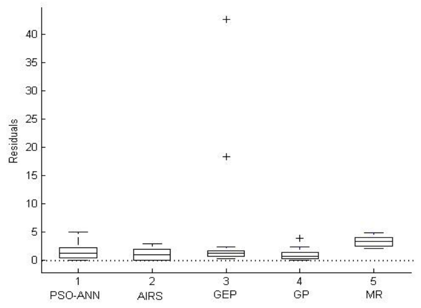
(b) Plot of predicted vs. the actual FST values at the unit testing phase for techniques having significant goodness of fit.



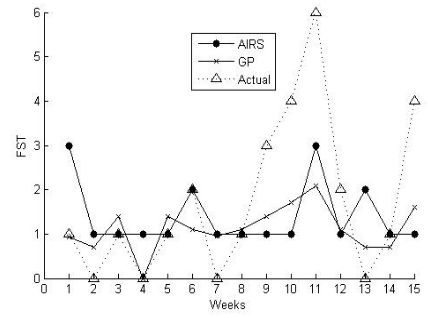
(c) Box plots of the residuals for each technique at the function testing phase.



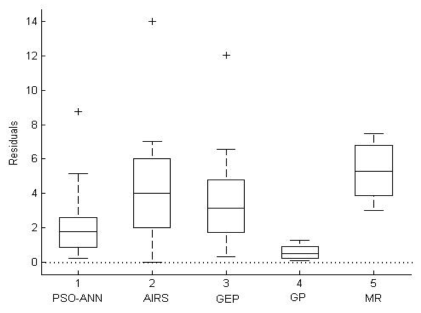
(d) Plot of predicted vs. the actual FST values at the function testing phase for techniques having significant goodness of fit.



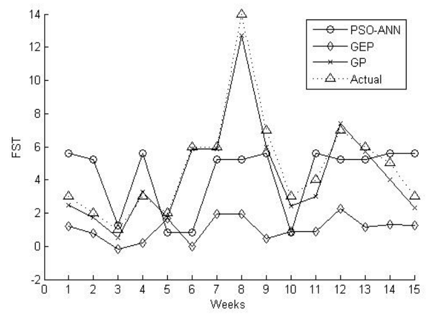
(e) Box plots of the residuals for each technique at the integration testing phase.



(f) Plot of predicted vs. the actual FST values at the integration testing phase for techniques having significant goodness of fit.



(g) Box plots of the residuals for each technique at the system testing phase.



(h) Plot of predicted vs. the actual FST values at the system testing phase for techniques having significant goodness of fit.

Figure 2. Box plots of the residuals and the plot of predictions for unit, function, integration and system testing phase.

Table V

Two-sample two sided K-S test results for predictions at different testing phases with critical value $J_\alpha$ = 0.54.

| *K-S test statistic, J* | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Unit testing | | | | | Function testing | | | | | Integration testing | | | | | System testing | | | | |
| PSO-ANN | AIRS | GEP | GP | MR | PSO-ANN | AIRS | GEP | GP | MR | PSO-ANN | AIRS | GEP | GP | MR | PSO-ANN | AIRS | GEP | GP | MR |
| 0.8 | 0.27 | 0.8 | 0.6 | 1 | 0.40 | 0.90 | 0.90 | 0.90 | 1 | 0.60 | 0.27 | 0.60 | 0.33 | 0.73 | 0.40 | 0.73 | 0.27 | 0.20 | 0.67 |

Table VII

p-values after applying the Wilcoxon rank sum test on residuals at unit, function, integration and system testing phases with $\alpha$ = 0.05.

| *Unit testing* | | | | *Function testing* | | | | | | | | *Integration testing* | | | | *System testing* | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MR:PSO-ANN | MR:AIRS | MR:GEP | MR:GP | PSO-ANN:AIRS | PSO-ANN:GEP | PSO-ANN:GP | AIRS:GEP | AIRS:GP | MR:PSO-ANN | MR:GEP | MR:GP | MR:PSO-ANN | MR:AIRS | MR:GEP | MR:GP | PSO-ANN:AIRS | PSO-ANN:GP | AIRS:GP | GEP:GP | MR:PSO-ANN | MR:GEP | MR:GP |
| 0.00004 | 0.000003 | 0.000003 | 0.000002 | 0.00 | 0.04 | 0.04 | 0.03 | 0.03 | 0.0014 | 0.008 | 0.006 | 0.0004 | 0.00009 | 0.0008 | 0.00006 | 0.04 | 0.00 | 0.00 | 0.00 | 0.0002 | 0.02 | 0.000003 |

test statistic being less than the critical value of $J_\alpha = 0.54$ (Figure 2(d)). As for ARE, GP has the lowest ARE value in comparison with other techniques (Table VI).

The residual box plots from the five techniques for predicting FST at the *system testing phase* are shown in Figure 2(g). As with the box plots for the FST at the function testing phase (Figure 2(c)), there is a greater variance for different techniques at the system testing phase (Figure 2(g)). The sizes of the box plots vary, with AIRS having a larger box plot in comparison with the others. The box plot for GP is smallest with the median closer to 0. To test for any significant differences between the residuals, the results of applying the Kruskal-Wallis test ($p = 0.0000001$) suggested that the null hypothesis of all the samples being drawn from the same population can be rejected at $\alpha = 0.05$. We subsequently used the Wilcoxon rank sum test as a post-hoc test to determine which particular comparisons differ significantly. The $p$-values obtained are shown in Table VII. The results indicate that there are significant differences in residuals between the pairwise combinations of PSO-ANN:GP, AIRS:GP, GEP:GP, PSO-ANN:AIRS, MR:PSO-ANN, MR:GEP, MR:GP (having a $p$-value of less than 0.05). A common trend from this result show that the predictions made by GP are significantly different from the others, a trend that is also confirmed from the box plots in Figure 2(g). The goodness of fit of predictions from each of the techniques with actual FST data during system testing phase is measured using the K-S test. The results appear in Table V. The results in Table V show that PSO-ANN, GEP and GP have statistically significant goodness of fit (Figure 2(h)). As is evident from the box plots in Figure 2(g), AIRS has a wider box plot and consequently has a non-significant goodness of fit in relation to the actual FST at the system testing phase. ARE values from different techniques (Table VI) indicate that GP gives the lowest ARE values.

In summary, a general trend from the results show that the search-based techniques (PSO-ANN, AIRS, GEP, GP) perform better than multiple regression for predicting FST at unit, function, integration and system testing phases. The results show that MR model residuals are different and inferior for majority of the pair-wise comparisons with search-based techniques' residuals for all the testing phases. The goodness of fit of MR is not significant and ARE values not the lowest for any of the testing phases in comparison with search-based techniques. At the *unit* testing phase the model residuals for four search-based techniques do not differ significantly but AIRS performance is better in terms of having both statistically significant goodness of fit and lowest ARE value. At the *function* testing phase the box plots of model residuals for the search-based techniques show a certain degree of variability but PSO-ANN box plot is promising with its median nearer to the 0 mark on $y$-axis. PSO-ANN is also better in having both statistically significant goodness of fit and lowest ARE value. At the *integration* testing phase, the performance of GP is better with having the median residuals closer to the 0 mark on $y$-axis, statistically significant goodness of fit and lowest ARE value; while at the *system* testing phase GP is again better in terms of having significantly different and better residuals, goodness of fit and lowest ARE.

So while MR is not able to perform as good as the search-based techniques, GP performs better for two testing phases (integration and system) and AIRS, PSO-ANN perform better for unit and function testing phases respectively.

## VII. Discussion

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

An overall interesting outcome of the study is that a variety of search-based techniques perform better than multiple regression for FST prediction, especially GP performs better for two of the four testing phases. The search-based techniques are also likely to be robust to changes in the process since they are assumption-free and do not require prior definition of the functional structure to evolve.

While the focus of this paper has been on a quantitative evaluation of techniques, one has to be mindful that there are additional qualitative characteristics of the empirical models that are important for real-world use, e.g. lower cost of ownership and robustness to withstand minor process changes. While these qualitative issues are not the focus in this study, they are still worth investigating e.g. by asking industrial professionals to fill structured questionnaires to assess the usefulness and usability aspects. Additionally building automated tool support for different search-based techniques would ease parameter tuning and would allow the practitioners to evaluate the results of applying multiple techniques more easily. Moreover selection of predictor variables that are easy to gather (e.g. the project level metrics at the subject company in this study) and that do not conflict with the development life cycle have better chances of industry acceptance. There is evidence to support that general process level metrics are more accurate than code/ structural metrics [35], however this subject requires further research.

Another interesting outcome of this study is the performance of search-based techniques outside their respective training ranges, i.e., the predictions are evaluated for 15 weeks of an on-going project after being trained on another baseline project data. This is to say that the over-fitting is within acceptable limits that can be judged from the ARE values in Table VI where the values for GP are between $0.04$ and $3.15$ and for PSO-ANN between $0.43$ and $2.66$; they are acceptable not being only smaller but also considering the fact that we are dealing with large scale projects where the degree of variability in fault occurrences can be large. This issue is also related to the amount of data available for training the different techniques which in case of large projects is typically available enough for the prediction techniques. We used the historical data from the past 45 weeks to train the models.

In short the implication of the results in this study is that they have the potential to provide *early* indications of quantified improvement potential within the software testing life cycle.

## VIII. Validity evaluation

*Conclusion validity* refers to a statistically significant relationship between the treatment and the outcome. We have used non-parametric statistics in this study due to the violation of the normality assumptions required for parametric statistical tests. Since the power of parametric statistical tests are generally higher than non-parametric ones, there is a chance that the use of non-parametric statistics might have missed potential differences in outcomes. However this threat is minimized to an extent by additionally looking at the box-plots where general trends are visible. *Internal validity* refers to a causal relationship between treatment and outcome. A possible threat to internal validity is that we cannot publicize our industrial data sets due to proprietary concerns. However, the transformed representation of the data can be made available if requested. Secondly each of the search-based techniques are executed 10 times to minimize the effect of randomness inherent in these techniques. *Construct validity* is concerned with the relationship between theory and application. Our choice of selecting project level metrics (Table I) instead of structural code metrics was influenced by multiple factors. First, metrics relevant to work packages (Section IV) have an intuitive appeal for the employees at the subject company where they can relate FST to the proportion of effort invested. Secondly, the existence of a module in multiple work packages made it difficult to obtain consistent metrics at the component level. Thirdly, the intent of this study is to use project level metrics that are readily available and hence reduces the cost of doing such predictions. *External validity* is concerned with generalization of results outside the scope of the study. This study is designed for projects structured according to work packages (Section III), therefore a different set of variables are required for projects structured differently. However the use of project level metrics as predictors of fault-slippages require more empirical studies to increase generalizability, e.g., by using more projects from different domains.

## IX. Conclusions

This paper has evaluated the application of five techniques for predicting the fault-slippage in various testing phases using two projects from the telecommunication industry. One of the projects is used as a baseline project to train the techniques while the other on-going project is used to evaluate the prediction performance of each of the techniques. Various project level metrics are used for building the techniques. The results from the empirical study show that the search-based techniques (PSO-ANN, AIRS, GEP, GP) perform better than multiple regression for predicting FST at different testing phases based on model residuals, goodness of fit and absolute relative error values. At the unit testing phase, though there are no significant differences between the model residuals of the search-based techniques, AIRS show statistically significant goodness of fit and lowest ARE value. PSO-ANN perform better at function testing phase – having residuals more closer to 0, statistically significant goodness of fit and lowest ARE value. At integration and system testing phases, GP perform better with respect to the three measures of model residuals, goodness of fit and absolute relative error values. We conclude that a variety of search-based techniques are applicable for predicting fault-slippage between different testing phases, especially the use of GP as a prediction technique is promising as it shows better prediction performance for two of the four testing phases.

## References

[1] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Tran. on SW Eng.*, vol. 25, no. 5, 1999.

[2] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Computer*, vol. 34, no. 1, 2001.

[3] L.-O. Damm, L. Lundberg, and C. Wohlin, "Faults-slip-through–A concept for measuring the efficiency of the test process," *SW Process: Improv. & Prac.*, vol. 11, no. 1, 2006.

[4] L.-O. Damm, "Early and cost-effective software fault detection – measurement and implementation in an industrial setting," Ph.D. dissertation, Blekinge Inst. of Tech., 2007.

[5] M. Staron and W. Meding, "Predicting weekly defect inflow in large software projects based on project planning and test status," *IST*, vol. 50, no. 7–8, 2008.

[6] S. Rakitin, *Software verification and validation for practitioners and managers*, 2nd ed. Artech House., Inc., 2001.

[7] E. K. Burke and G. Kendall, Eds., *Search methodologies – Introductory tutorials in optimization and decision support techniques*. Springer Science and Business Media, Inc., 2005.

[8] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *ICSE'05*, 2005.

[9] L. Briand, K. Emam, B. Freimut, and O. Laitenberger, "A comprehensive evaluation of capture-recapture models for estimating software defect content," *IEEE Tran. on SW Eng.*, vol. 26, no. 6, 2000.

[10] J. Munson and T. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Tran. on SW Eng.*, vol. 18, no. 5, 1992.

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

[12] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Tran. on SW Eng.*, vol. 31, no. 10, 2005.

[13] A. Veevers and A. C. Marshall, "A relationship between software coverage metrics and reliability," *Software Testing, Verification and Reliability*, vol. 4, no. 1, 1994.

[14] V. Challagulla, F. Bastani, I. Yen, and R. Paul, "Empirical assessment of machine learning based software defect prediction techniques," in *Proc. of the 10th IEEE Workshop on OO Real-Time Dependable Systems*, 2005.

[15] W. Afzal and R. Torkar, "A comparative evaluation of using genetic programming for predicting fault count data," in *Proc. of the 3rd int. conf. on SW eng. advances*. IEEE, 2008.

[16] C. Catal and B. Diri, "Software fault prediction with object-oriented metrics based AIRS," in *PROFES'07*. LNCS, 2007.

[17] S. Canu, Y. Grandvalet, V. Guigue, and A. Rakotomamonjy, "SVM and kernel methods toolbox," Perception Systémes et Information, INSA de Rouen, Rouen, France, 2005.

[18] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd, "What accuracy statistics really measure?" *IEE Proceedings Software*, vol. 148, no. 3, 2001.

[19] M. Shepperd, M. Cartwright, and G. Kadoda, "On building prediction systems for software engineers," *Empirical Software Engineering*, vol. 5, no. 3, 2000.

[20] T. Khoshgoftaar, N. Seliya, and N. Sundaresh, "An empirical study of predicting software faults with CBR," *Software Quality Control*, vol. 14, no. 2, 2006.

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

[22] G. Zhang, "Avoiding pitfalls in neural net. research," *IEEE Tran. on Systems Man and Cybernetics*, vol. 37, no. 1, 2007.

[23] B. Curry and P. Morgan, "Neural networks: A need for caution," *Omega*, vol. 25, no. 1, 1997.

[24] G. K. Jha, P. Thulasiraman, and R. K. Thulasiram, "PSO based neural network for time series forecasting," in *International Joint Conference on Neural Networks*, 2009.

[25] J. Kennedy and R. Eberhart, "PSO," in *Proc. of the IEEE Int. Conf. on Neural Networks*, 1995.

[26] I. C. Trelea, "The PSO algorithm: convergence analysis and parameter selection," *IP Letters*, vol. 85, no. 6, 2003.

[27] B. Birge, "PSO–MATLAB toolbox," 2005, http://www.mathworks.com/matlabcentral/fileexchange/7506-particle-swarm-optimization-toolbox.

[28] A. Watkins, J. Timmis, and L. Boggess, "Artificial immune recognition system (AIRS): An immune-inspired supervised learning algorithm," *GPEM*, vol. 5, no. 3, 2004.

[29] J. Timmis, M. Neal, and J. Hunt, "An artificial immune system for data analysis," *Biosystems*, vol. 55, no. 1-3, 2000.

[30] J. Brownlee, "WEKA plug-in for AIRS." http://wekaclassalgos.sourceforge.net/, 2010.

[31] C. Ferreira, "GEP: A new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, 2001.

[32] P. H. Sherrod, *DTREG*, Predictive modeling and forecasting, 2010, http://www.dtreg.com/DownloadManual.htm.

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

[34] S. Silva, "GPLAB—A genetic programming toolbox for MATLAB," http://gplab.sourceforge.net.

[35] E. Arisholm, L. Briand, and E. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *JSS*, vol. 83, no. 1, 2010.