Software Qual J
DOI 10.1007/s11219-013-9205-3

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

Wasif Afzal · Richard Torkar · Robert Feldt · Tony Gorschek

Abstract A large percentage of the cost of rework can be avoided by finding more faults earlier in a software test process. Therefore, determination of which software test phases to focus improvement work on has considerable industrial interest. We evaluate a number of prediction techniques for predicting the number of faults slipping through to unit, function, integration, and system test phases of a large industrial project. The objective is to quantify improvement potential in different test phases by striving toward finding the faults in the right phase. The results show that a range of techniques are found to be useful in predicting the number of faults slipping through to the four test phases; however, the group of search-based techniques (genetic programming, gene expression programming, artificial immune recognition system, and particle swarm optimization–based artificial neural network) consistently give better predictions, having a representation at all of the test phases. Human predictions are consistently better at two of the four test phases. We conclude that the human predictions regarding the number of faults slipping through to various test phases can be well supported by the use of search-based techniques. A combination of human and an automated search mechanism (such as any of the search-based techniques) has the potential to provide improved prediction results.

Keywords Prediction · Empirical · Faults-slip-through · Search-based

W. Afzal (✉)
Department of Computer Sciences, Bahria University, Shangrilla Road, Sector E-8,
Islamabad 44000, Pakistan
e-mail: wasif.afzal@gmail.com

R. Torkar · R. Feldt
Department of Computer Science and Engineering, Chalmers University of Technology,
41296 Gothenburg, Sweden
e-mail: richard.torkar@chalmers.se; richard.torkar@bth.se

R. Feldt
e-mail: robert.feldt@chalmers.se; robert.feldt@bth.se

R. Torkar · R. Feldt · T. Gorschek
School of Computing, Blekinge Institute of Technology, 37179 Karlskrona, Sweden
e-mail: tony.gorschek@bth.se

Published online: 15 May 2013

1 Introduction and problem statement

Presence of a number of faults[1] usually indicates an absence of software quality. Software testing is the major fault-finding activity; therefore, much research has focused on making the software test process as efficient and as effective as possible. One way to improve the test process efficiency is to avoid unnecessary rework by finding more faults earlier. This argument is based on the premise that the faults are cheaper to find and remove earlier in the software development process (Boehm and Basili 2001). Faults-slip-through (FST) metric (Damm et al. 2006; Damm 2007) is one way of providing quantified decision support to reduce the effort spent on rework.

Faults-slip-through (FST) metric is used for determining whether a fault slipped through the phase where it should have been found or not (Damm et al. 2006; Damm 2007). The term phase refers to any phase in a typical software development life cycle (ISO/ IEC 12207 (STD 2008) defines the different software development phases). However, the most interesting and industry-supported applications of FST measurement are in the test phase of a software development life cycle, because it is typically in this phase where the faults are classified into their actual and expected identification phases.

The time between when a fault was inserted and found is commonly referred to as 'fault latency' (2008). Figure 1 shows the difference between fault latency and FST (Damm et al. 2006; Damm 2007). As it is clear from this figure, the FST measurement evaluates when it is cost efficient to find a certain fault. To be able to specify this, the organization must first determine what should be tested in which phase (Damm et al. 2006; Damm 2007).

Studies on multiple projects executed within several different organizations at Ericsson (Damm 2007) showed that FST measurement has some promising advantages:

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

Figure 2 shows an example snippet of a faults-slip-through matrix showing the faults slipping through to later phases. The columns in Fig. 2 represent the phases in which the faults were found (*Found During*), whereas the rows represent the phases where the faults should have been found (*Expected fault identification phase*). For example, 56 of the faults that were found in the function test should have been found during the unit test.

Apart from the studies done by Damm et al. (2006) and Damm (2007), there are other studies on successful industrial implementation of FST measurements. Two such cases are the FST implementations at Ericsson Nikola Tesla (Hribar 2008; Antolić 2007). They started collecting FST measurements in all development projects from the middle of the year 2006. The results were encouraging with a decrease in fault-slippage to customers, improvements of test configurations, and improvements of test cases used in the verification phase of the projects.

Considering the initial successful results of implementing FST measurement across different organizations within Ericsson, our industrial partner became interested in investigating how to use FST measurement to provide additional decision support for project management. For example, Staron and Meding (2008) highlight that the prediction

[1] According to IEEE Standard Glossary of Software Engineering Terminology (IEEE 1990), a fault is a manifestation of a human mistake.

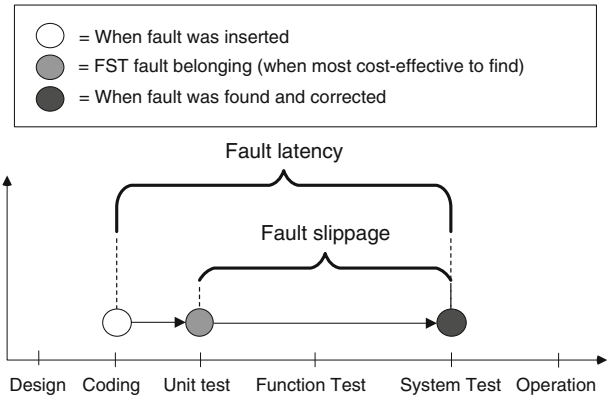


Fig. 1 Difference between fault latency and FST

of the number of faults slipping through can be a refinement to their proposed approach for predicting the number of defects in the defect database. Similarly, Damm (2007) highlights that FST measurement can potentially be used as a support tool in software fault predictions. This additional decision support is to make the software development more predictable (Rakitin 2001).

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

In this paper, we focus on predicting the number of faults slipping through to different test phases, multiple weeks in advance (a quantitative modeling task). We compare a variety of prediction techniques[2]. Since there is a general lack of empirical evaluation of expert judgement (Tomaszewski et al. 2007; Catal and Diri 2009) and due to the fact that predictions regarding software quality are based on expert judgements at our organization, and we would argue in industry in general, we specifically compare human expert predictions with these techniques. Thus, the motivation of doing this study is to:

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

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

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

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

Fig. 2 An example FST matrix

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

RQ.1 How do different techniques compare in FST prediction performance?
RQ.2 Can other techniques better predict the number of faults slipping through to different test phases than human expert judgement?

The data used in the quantitative data modeling come from large and complex software projects from the telecommunications industry, as our objective is to come up with results that are representative of real industrial use. Also large-scale projects offer different kinds of challenges, e.g., the factors affecting the projects are diverse and many, data are distributed across different systems, and success is dependent on the effort of many resources. Moreover, a large project constitutes a less predictable environment, and there is a lack of research on how to use predictive models in such an environment (Jørgensen et al. 2000).

We also would like to mention that this study is an extended version of the authors' earlier conference manuscript (Afzal et al. 2010) where only a limited number of techniques were compared with no evaluation of expert judgement.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 describes the study context, variables selection, the test phases under consideration, the performance evaluation measures, and the techniques used. Section 4 presents a quantitative evaluation of various techniques for the prediction task. The results from the quantitative evaluation of different models are discussed in Sect. 5, while the study validity threats are given in Sect. 6. The paper is concluded in Sect. 7, while "Appendix" outlines the parameter settings for the different techniques.

2 Related work

Due to the definition of software quality in many different ways, previous studies have focused on predicting different but related dependent variables of interest; examples include predicting for defect density (Nagappan and Ball 2005; Mohagheghi et al. 2004), software defect content estimation (Briand et al. 2000; Weyuker et al. 2010), fault-proneness (Lessmann et al. 2008; Arisholm et al. 2010), and software reliability prediction in terms of time-to-failure (Lyu 1996). In addition, several independent variables have been used to predict the above dependent variables of interest; examples include prediction using size and complexity metrics (Gyimothy et al. 2005), testing metrics (Veevers and Marshall 1994; Tomaszewski et al. 2007), and organizational metrics (Nagappan et al. 2008). The actual prediction is performed using a variety of approaches and can broadly be

Table 1 Variables of interest for the prediction models

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

classified into statistical regression techniques, machine-learning approaches, and mixed algorithms (Challagulla et al. 2005). Increasingly, evolutionary and bio-inspired approaches are being used for software quality classification (Liu et al. 2010; Afzal and Torkar 2008), while expert judgement is used in very few studies (Tomaszewski et al. 2007; Zhong et al. 2004).

For a more detailed overview of related work on software fault prediction studies, the reader is referred to Tian (2004), Fenton and Neil (1999), Catal and Diri (2009), Wagner (2006), and Runeson et al. (2006).

This study is different from the above software quality evaluation studies. First, the dependent variable of interest for the quantitative data modeling is the number of faults slipping through to various test phases, with the aim of taking corrective actions for avoiding unnecessary rework late in software testing. Second, the independent variables of interest for the quantitative data modeling are diverse and at the *project level*, i.e., variables depicting work status, testing progress status and fault-inflow (shown later in Table 1). A similar set of variables were used in a study by Staron and Meding (2008), but predicted weekly defect inflow and used different techniques. Third, for the sake of comparison, we include a variety of carefully selected techniques, representing both commonly used and newer approaches.

Together this means our study is broader and more industrially relevant than previous studies.

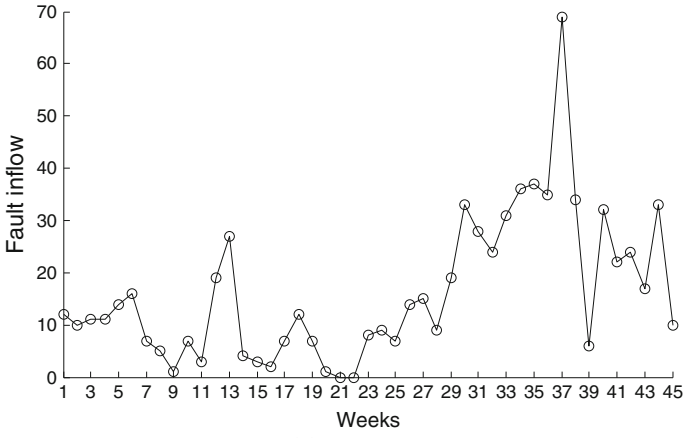
3 Study plan

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

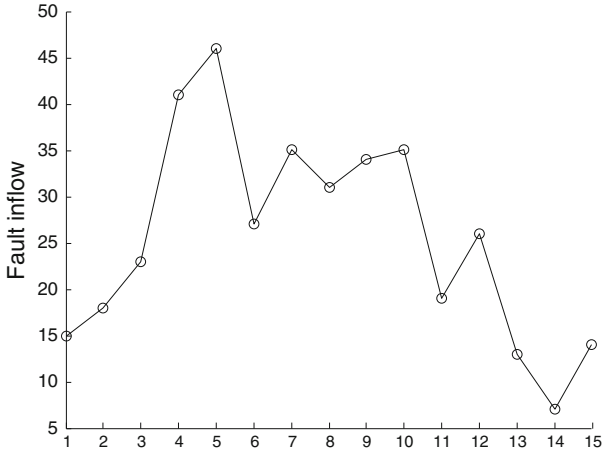
3.1 Study context

As given in Sect. 1, our context is large and complex software projects in the telecommunications industry. Our subject company develops mobile platforms and wireless semiconductors. The projects are aimed at developing platforms introducing new radio access technologies written using the C programming language. The average number of persons involved in these projects is approximately 250. Since the project is from a similar domain, the data from one of the projects are used as a baseline to train the models, while the data from the second project are 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 ongoing project. Figure 3 shows the number of faults occurring per week for the training and the testing set.

The management of these projects follow the company's general project model called PROfessional Project Steering (PROPS). PROPS is based on the concepts of tollgates,



(a) Training set.



(b) Testing set.

Fig. 3 Number of fault occurrences per week for the training set and the testing set

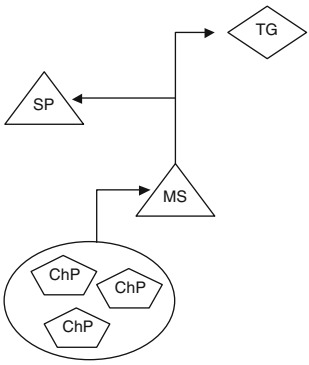


Fig. 4 PROPS concepts used in the subject company; TG, SP, MS, ChP are short for tollgate, steering point, milestone, and checkpoint, respectively

milestones, steering points, and checkpoints to manage and control project deliverables. Tollgates represent long-term business decisions, while milestones are predefined events representing intermediate objectives at the operating work level. The monitoring of these milestones is an important element of the project management model. Steering points are defined to coordinate multiple parallel platform projects, e.g., handling priorities between different platform projects. The checkpoints are defined in the development process to define the work status in a process. Multiple checkpoints might have to be passed for reaching a certain milestone. Figure 4 shows an abstract level view of these concepts.

At the operative work level, the software development is structured around *work packages*. 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 it allows 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 presents an obvious choice of selecting variables for the prediction models since the metrics at work package level are stable and entail a more intuitive meaning for the employees at the subject company.

Figure 5 gives an overview of how a given project is divided into work packages that affect multiple modules. There are three sub-systems shown in Fig. 5, namely A, B, and C. The division of an overall system into sub-systems is driven by design and architectural constraints. The modules belonging to the three sub-systems are named as (A1, A2), (B1, B2), and (C1, C2, C3), respectively. The overall project is divided into a number of work packages which are named in Fig. 5 as WP1, WP2,…, WPn. Since changes can be made to multiple modules when developing a single work package, this is shown as dashed arrow lines. The division of a project into work packages is more definitive with clear boundaries; therefore, this division is shown as solid arrow lines. Figure 5 also shows the checkpoints, steering points, and the tollgates that are meant to manage and control project deliverables.

3.2 Variables selection

At our subject company, the work status of various work packages is grouped using a graphical integration plan (GIP) document. The GIP maps the work packages' status over multiple time lines that might indicate different phases of software testing or overall

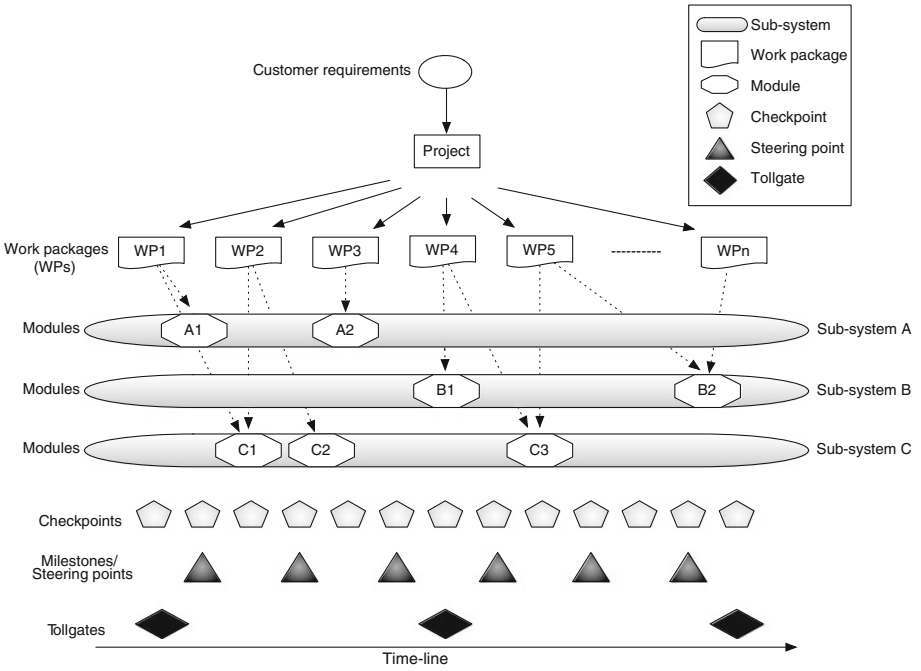


Fig. 5 Division of requirements into work packages and modules (work model), thereby achieving tollgates, milestones/steering points, and checkpoints (management model)

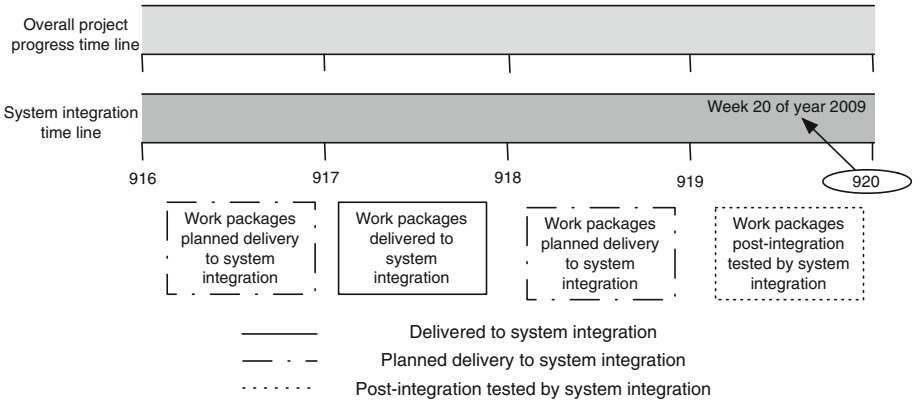


Fig. 6 The graphical integration plan showing the status of various work packages over multiple time lines

project progress. There are different *status rankings* of the work packages, e.g., number of work packages planned to be delivered for system integration testing. A snippet of a GIP is shown in Fig. 6.

The variables of interest in this study are divided into four sets (Table 1), i.e., fault-inflow, status rankings of work packages, faults-slip-through, and test case progress. A description of these four sets of variables is given below.

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 1) 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 1, which have a more direct influence on the fault-inflow. The information on the number of test cases planned and executed at different test phases is readily available from an automated report generation tool that uses data from an internally developed system for fault logging. These variables, along with the status rankings of the work packages, influence the fault-inflow; so we monitor the fault-inflow as another variable for our prediction models. Another set of variables representing the output is the number of faults that slipped through to the unit, function, integration, and system test phases, indicated under the category 'FST' in Table 1. We also recorded the accumulated number of faults slipping through to all the test phases. All of the above measurements were collected at the subject company on a weekly basis.

3.3 Test phases under consideration

Software testing is usually performed at different levels, i.e., at the level of a single module, a group of such modules or a complete system (swe 2006). These different levels are termed as *test phases* in our subject company; therefore, we stick to calling them test phases throughout the paper. The purpose of different test phases, as defined at our subject company, is given below:

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

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

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

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

3.4 Performance evaluation measures and prediction techniques

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

- The predictive accuracy of different techniques is compared using absolute residuals (i.e., |actual-predicted|) (Pickard et al. 1999; Kitchenham et al. 2001; Shepperd et al. 2000).

– The goodness of fit of the results from different techniques is assessed using the two-sample two-sided 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_{α}, we infer that the two samples did not have the same probability distribution and hence do not represent significant goodness of fit.

We consider a technique better based on the following criteria

– If technique A performs statistically significantly better than technique B for both predictive accuracy and goodness of fit, then technique A is declared as better.
– If no statistically significant differences are found between techniques A and B for predictive accuracy, but technique B has statistically significant goodness of fit in comparison with technique A, then technique B is declared as better.
– If no statistically significant differences are found between techniques A and B for goodness of fit, but technique B has statistically significant predictive accuracy in comparison with technique A, then technique B is declared as better.

The above-mentioned evaluation procedure is an example of multi-criteria-based evaluation system, a concept similar to the one presented by Lavesson and Davidsson in (2008).

A brief description of each of the different techniques used in this study appears in Table 2

4 Analysis and interpretation

This section describes the quantitative analysis helping us answer our research questions.

4.1 Analyzing dependencies among variables

Before applying the specific techniques for prediction, we analyzed the dependencies among variables (see Table 1) using scatter plots. We were especially interested in visualizing:

– The relationship between the measures of status rankings of work packages.
– The relationship between the measures of test case progress.
– Fault-inflow versus the rest of the measures related to status rankings of work packages and test case progress.

The pairwise scatter plots of the above attributes showed a tendency of nonlinear relationship. Two of these scatter plots are shown in Fig. 7 for *fault-inflow* versus *number of faults slipping through all of the test phases* (Fig. 7a) and *fault-inflow* versus *number of work packages tested by system integration* (Fig. 7b).

After getting a sense of the relationships among the variables, we used kernel principal component analysis (KPCA) (Canu et al. 2005) 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 3) 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

Table 2 Prediction techniques used in this study

Prediction technique	Brief overview
Statistical techniques	
Multiple regression (MR), Pace regression (PR)	MR is an extension of simple least-square regression for more than one independent (predictor) variables to estimate the values of the dependent (criterion) variable. More information on MR is available in (Kachigan 1982). PR decomposes an initially estimated model (the ordinary least-square estimator) into statistically independent dimensional models. These dimensional models are then used to obtain an estimate of the true effects in each individual dimension. More details on PR in (Wang 2000).
Tree-structured techniques	
M5P, REPTree	M5P is tree induction for regression models. A decision tree induction algorithm is used to build a tree, but instead of maximizing the information gain at each inner node, a splitting criterion is used that minimizes the intra-subset variation in the class values down each branch. More information on M5P is available in (Wang and Witten 1996). REPTree builds a decision/regression tree using the information gain/variance and prunes it using reduced-error pruning with back-fitting. More information on REPTree is available in Weka documentation at (Weka Documentation 2010).
Nearest neighbor techniques	
K-Star (K*), *K*-nearest neighbor (Knn)	Both K* and Knn techniques are analogy-based methods which considers K most similar examples for performing classification. The similarity is measured in K* using an entropy-based distance function (more information in (Cleary and Trigg 1995) while an euclidean distance is used in knn (more information in (Aha et al. 1991))
Ensemble techniques	
Bagging, rotation forest (RF)	Ensemble techniques include several base learners and a voting procedure is used for final classification and an average for a numeric prediction. Bagging generates multiple versions of a predictor and an aggregated average over versions give a numeric outcome. More information in (Breiman 1996). RF splits the feature set into k-subsets and principal component analysis is applied to each subset. K-axis rotation forms new features for the base learner. More information in (Rodriguez et al. 2006).
Machine-learning techniques	
Support vector machines (SVM), back-propagation artificial neural networks (ANN)	Support vector regression uses a SVM algorithm for numeric prediction. SVM 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 $\in$ (Witten and Frank, 2005). Apart from specifying $\in$, the upper limit C on the absolute value of the weights associated with each data point has to be enforced (known as capacity control). The 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 shows 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.

Table 2 continued

Prediction technique	Brief overview
Search-based techniques	
Genetic programming (GP), gene expression programming (GEP), artificial immune recognition systems (AIRS), particle swarm optimization based artificial neural network (PSO-ANN)	Search-based techniques model a problem in terms of an evaluation function and then uses a search technique to minimize or maximize that function. GP evolves tree-structured mathematical programs (called individuals) that are evaluated in a recursive manner. Diversity is introduced in programs through the operators of cross-over, mutation and reproduction. A solution is designated as the best solution after some iterations (generations) that maximize/minimize an evaluation function. More information about GP can be found in (Poli et al. 2008). In GEP, the individuals making-up the search space (called population) are encoded as linear strings of fixed length that are later expressed as nonlinear expression trees of different sizes and shapes. More information about GEP can be found in (Ferreira 2001). AIRS is inspired by the processes of vertebrate immune system, specifically how B and T lymphocytes improve their response to antigens over time (called affinity maturation). The details of the different steps of the AIRS algorithm can be found in (Watkins et al. 2004). PSO-ANN uses a particle swarm optimization (PSO) algorithm for training an ANN . PSO is inspired by the coordinated search of food by a swarm of birds. A swarm of particles move through a multidimensional search space for finding global optimum. Trelea (2003) proposed several improvements to a basic PSO, called Trelea I and Trelea II depending upon the values of parameters. The PSO version Trelea II is used in this study. More information on PSO-ANN is available at (Jha et al. 2009).
Expert judgement	An estimate/prediction is based on the experience of one or more people who are familiar with the development of software applications similar to that currently being predicted (Hughes 1996). The expert in this study has 20 years of work experience and currently holds the designation of systems verification process leader at our subject organization.

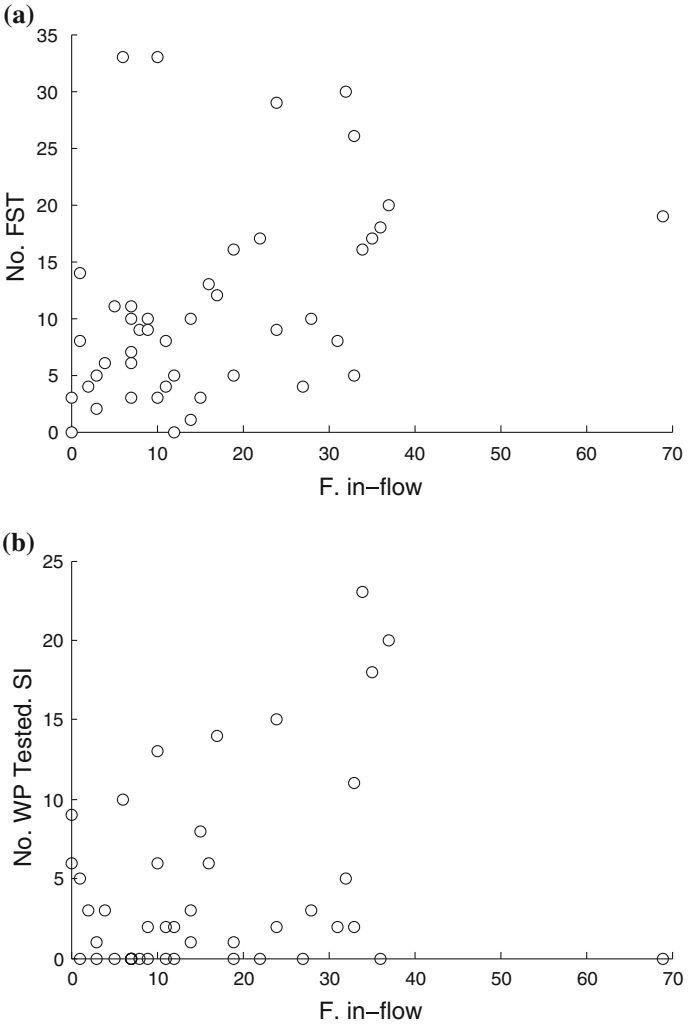


Fig. 7 Example *scatter plots* for fault-inflow versus number of faults slipping through all of the test phases and fault-inflow versus number of work packages tested by system integration. **a** *Scatter plot* for fault-inflow versus number of faults slipping through all of the test phases. **b** *Scatter plot* for fault-inflow versus number of work packages tested by system integration

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

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

Table 3 The loadings and explained variance from four principal components

	Variance explained (%)	Variable loadings										
		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

The variable names use abbreviations given in Table 1

the faults-slip-through at unit test phase as an independent variable along with fault-inflow, work-package status rankings, and test case progress metrics.

The model training and testing procedure along with the parameter settings for different techniques is given in detail in "Appendix".

4.2 Performance evaluation of techniques for FST prediction

Next, we present the results of the performance of different techniques in predicting FST for each test phase that would help us to answer RQ.1. As given in Sect. 3.4, we evaluate the prediction performance using the measures for predictive accuracy and goodness of fit.

The common analysis procedure to follow is to compare the box plots of the absolute residuals for different prediction techniques. But since box plots cannot confirm whether one prediction technique is significantly better than another, we use a statistical test (parametric or a nonparametric test—depending upon whether the assumptions of the test are satisfied) for testing the equality of population medians among groups of prediction techniques. Upon the rejection of the null hypothesis of equal population medians, a multiple comparisons (post-hoc) test is performed on the group medians to determine which means differ. Finally, we proceed with assessing the goodness of fit using the K–S test described in Sect. 3.4.

4.2.1 Prediction of FST at the unit test phase

The box plots of absolute residuals for predicting FST at the unit test phase for different techniques are shown in Fig. 8a. The box plot having the median value close to the 0 mark on the *y*-axis (shown as a dotted horizontal line in Fig. 8a) and a smaller spread of the distribution indicates better predictive accuracy. Keeping in view these two properties of the box plots, there seems to be only a marginal difference in the residual box plots of PR, M5P, Knn, Bagging, GEP, and PSO-ANN . AIRS has a median at the 0 mark but shows larger spread in comparison with other techniques. The human/expert prediction also shows a larger spread but smaller than AIRS . Two outliers for the human prediction are extreme as compared to the one extreme outlier for PSO-ANN . Predictions from MR, SVM, and ANN appear to be farther away from the 0 mark on the *y*-axis, an indication that the predictions are not closely matching the actual FST values.

To test for any statistically significant differences in the models' residuals, the non-parametric Kruskal–Wallis test was used to examine any statistical differences between the residuals and to confirm the trend observed from the box plots. The skewness in the residual box plots for some techniques motivated the use of the nonparametric test. The result of the Kruskal–Wallis test ($p = 3.2e^{-14}$) suggested that it is possible to reject the null hypothesis of all samples being drawn from the same population at significance level, $\alpha = 0.05$. This is to suggest that at least one sample median is significantly different from the others. In order to determine which pairs are significantly different, we apply a multiple comparisons test (Tukey–Kramer, $\alpha = 0.05$). The results of the multiple comparisons are displayed using a graph given in Fig. 8b. The mean of each prediction technique is represented by a circle, while the straight lines on both sides of the circle represents an interval. The means of two prediction techniques are significantly different if their intervals are disjoint and are not significantly different if their intervals overlap. For illustrative purposes, Fig. 8b shows vertical dotted lines for MR . There are two other techniques (SVM and ANN) where either of these two dotted lines cut through their intervals, showing that the means for MR, SVM, and ANN are not significantly different. It is interesting to

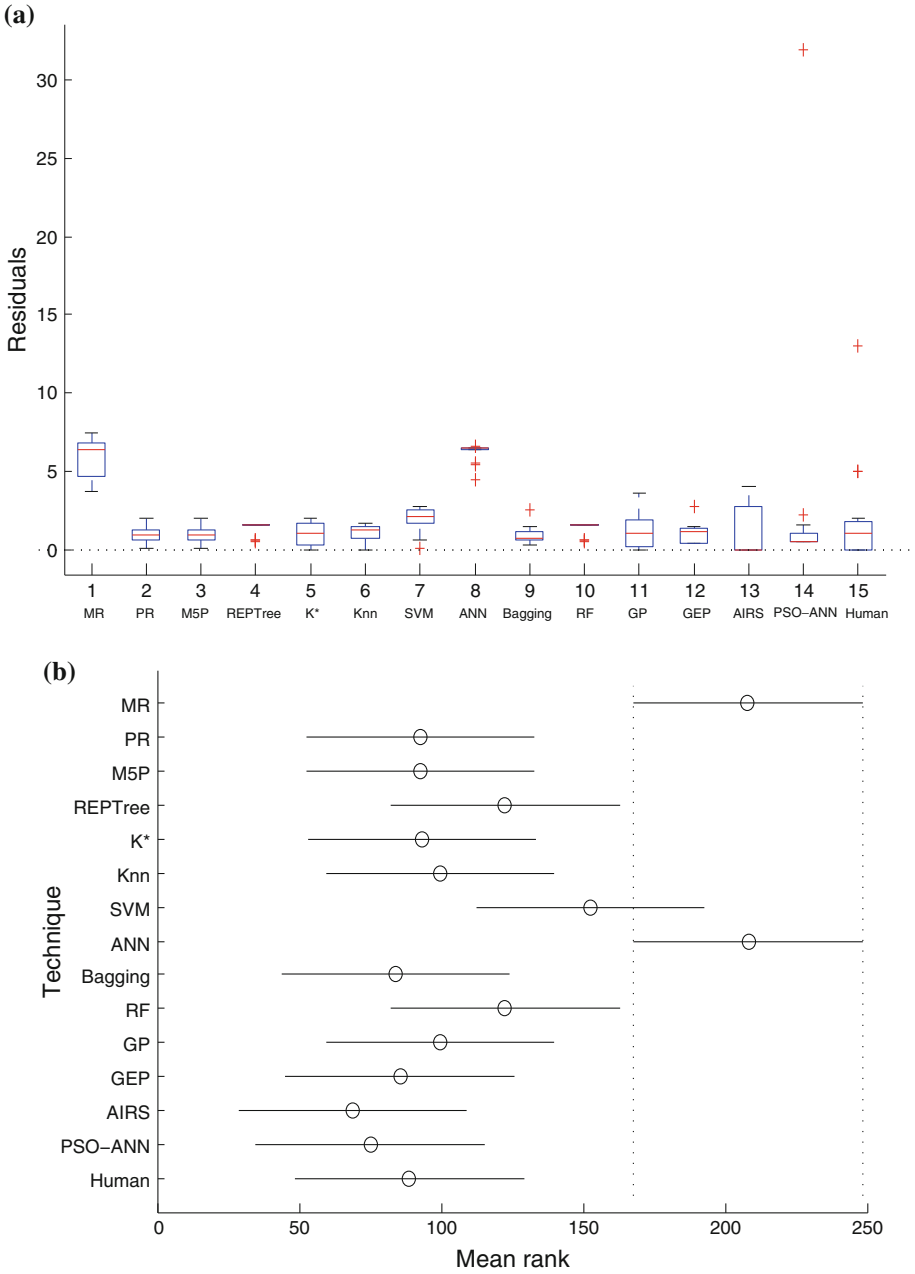


Fig. 8 Results showing *box plots* of absolute residuals and multiple comparisons of the absolute residuals between all techniques at the unit test phase. **a** *Box plots* of the residuals for each technique in predicting FST at the unit test phase. **b** Results of the multiple comparisons test with $\alpha = 0.05$ (the *vertical dotted lines* indicating that 12 techniques have mean ranks significantly different from MR)

observe that there is only a single technique (AIRS) whose mean is significantly different (and better) than all these three techniques (i.e., MR, SVM, and ANN). There are, however, no significant pairwise differences between the means of AIRS and rest of the techniques (i.e., PR, M5P, REPTree, K*, Knn, Bagging, RF, GP, GEP, PSO-ANN, Human). Human predictions, on the other hand, are significantly different and better than two of the least accurate techniques (MR, ANN).

The K–S test result for measuring the goodness of fit for predictions from each technique relative to the actual FST at the unit test phase appear in Table 4. The techniques having statistically significant goodness of fit are shown in bold (AIRS and Human). Figure 9 shows the plot of AIRS, human, and actual FST at the unit test phase. The statistically significant goodness of fit for AIRS and human can be attributed to the exact match of actual FST data on 9 out of 15 instances for AIRS and 5 out of 15 instances for the human. However, the human prediction is off by large values in the last 3 weeks that can also be seen as extreme outliers in Fig. 8a.

In summary, in terms of predictive accuracy, AIRS showed significantly different absolute residuals in comparison with the three least performing techniques for predicting FST at the unit test phase. But then there were found no significant differences between the absolute residuals of AIRS and rest of the 11 techniques. Human predictions showed significantly different absolute residuals in comparison with the two least performing

Table 4 Two-sample two-sided K–S test results for predicting FST at the unit test phase with critical value $J_{0.05} = 0.5$

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

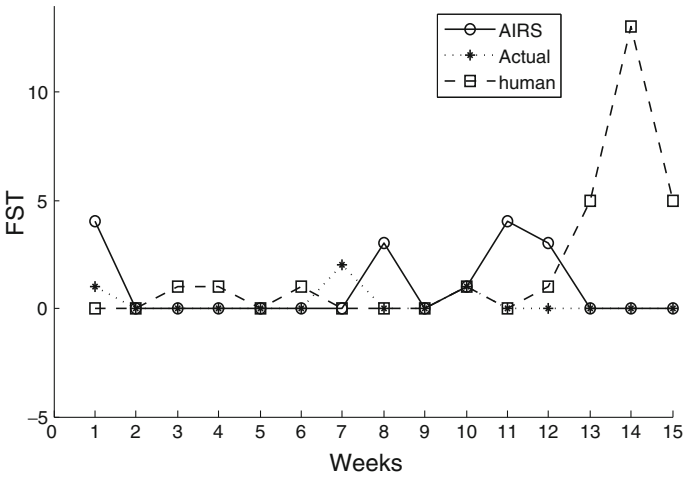


Fig. 9 Plot of the predicted versus the actual FST values at the unit test phase for techniques having significant goodness of fit

techniques for predicting FST at the unit test phase. For goodness of fit, AIRS and human predictions were found to be statistically significant, though the human predictions resulted in extreme values later in the prediction period.

4.2.2 Prediction of FST at the function test phase

The box plots of absolute residuals for predicting FST at the function test phase for different techniques are shown in Fig. 10a. We can observe that there is a greater spread of distribution for each of the techniques as compared with those at the unit test phase. The box plots for each of the techniques are also farther away from the 0 mark on the *y*-axis, with PSO-ANN and SVM having the median closet of all to the 0 mark on the *y*-axis. Human and MR prediction shows the greatest spread of distributions, while the box plots of PR, M5P, K*, Knn, and Bagging show only a marginal difference. The result of the Kruskal–Wallis test ($p = 1.6e^{-11}$) at $\alpha = 0.05$ suggested that at least one sample median is significantly different from the others. Subsequently, the results of the multiple comparisons test (Tukey–Kramer, $\alpha = 0.05$) appear in Fig. 10b. The absolute residuals of MR and human are not significantly different (as their intervals overlap), a confirmation of the trend observed from the box plots. Two of the better techniques having lower medians are SVM and PSO-ANN . There are no significant differences between the two. Also there are no significant pairwise differences between SVM and each one of: PR, M5P, REPTree, K*, Knn, Bagging, GP, GEP.

The K–S test result for measuring the goodness of fit for predictions from each technique relative to the actual FST at the function test phase appears in Table 5. SVM and PSO-ANN show statistically significant goodness of fit. Figure 11 shows the line plots of SVM and PSO-ANN with the actual FST at the function test phase. SVM appears to behave in Fig. 11 since there are no high peaks showing outliers (as is the case with PSO-ANN in Week 11).

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

4.2.3 Prediction of FST at the integration test phase

The box plots of absolute residuals for predicting FST at the integration test phase for different techniques are shown in Fig. 12a. We can observe that there is a smaller spread of distribution for each technique as compared with the box plots for function test. An exception is ANN whose box plot is more spread out than other techniques. In terms of the median being close to the 0 mark on the *y*-axis, Bagging and GP appear to be promising, though there seem to be only marginal differences in comparison with PR, M5P, REPTree, and PSO-ANN . GEP and human each shows two extreme outliers. The result of the Kruskal–Wallis test ($p = 1.7e^{-5}$) at $\alpha = 0.05$ suggested that at least one sample median is significantly different from the others. Subsequently, the results of the multiple comparisons test (Tukey–Kramer, $\alpha = 0.05$) appear in Fig. 12b. The mean rank for MR is not significantly different than the ones for SVM, ANN, and the human. GP has the mean rank that is significantly different than MR and ANN, the two least performing techniques.

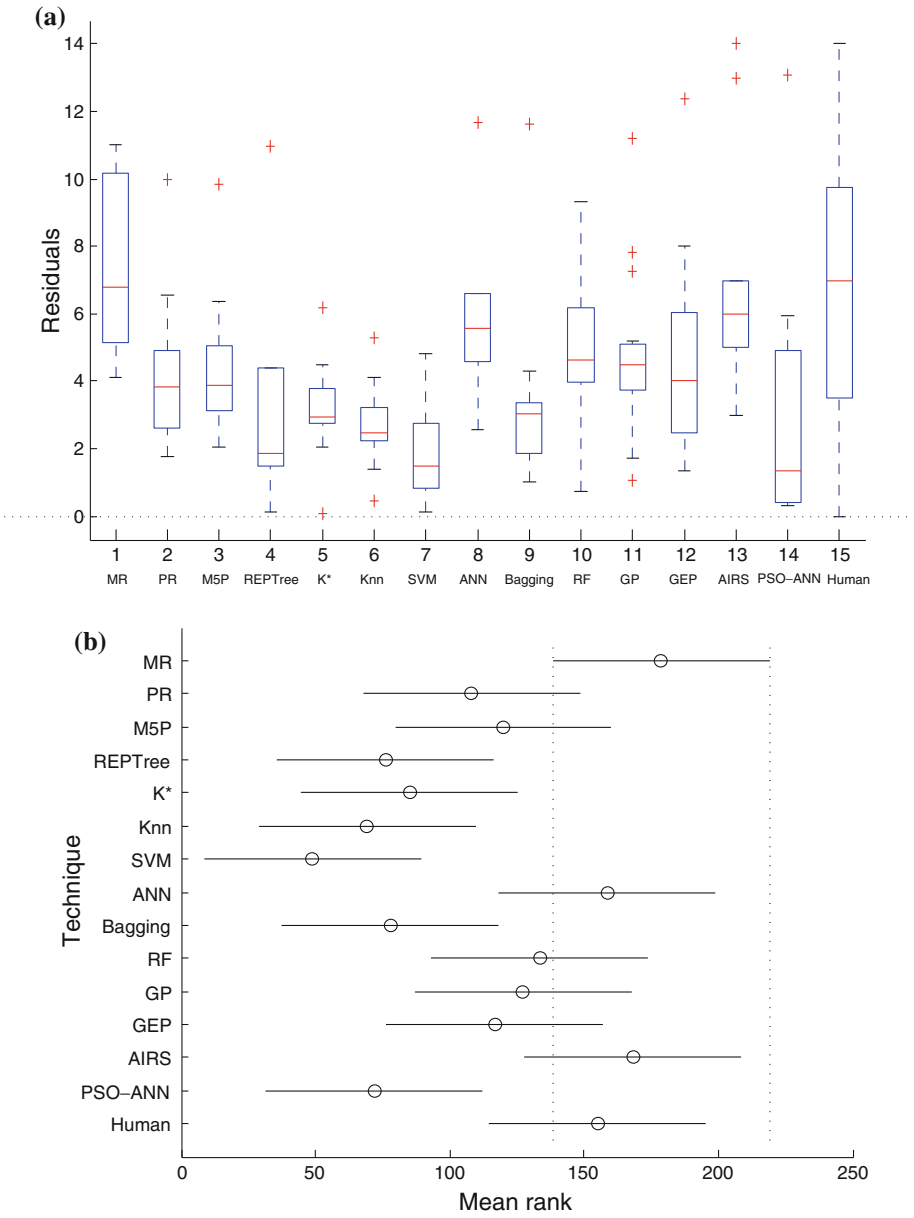


Fig. 10 Results showing box plots of absolute residuals and multiple comparisons of the absolute residuals between all techniques at the function test phase. **a** *Box plots* of the residuals for each technique in predicting FST at the function test phase. **b** Results of the multiple comparisons test with $\alpha = 0.05$ (the *vertical dotted lines* indicating that 6 techniques have mean ranks significantly different from MR)

However, there are not any pairwise significant differences between the absolute residuals for GP and each one of: PR, M5P, REPTree, K*, Knn, SVM, Bagging, RF, GEP, AIRS, PSO-ANN and Human.

Table 5 Two-sample two-sided K–S test results for predicting FST at the function test phase with critical value $J_{0.05} = 0.5$

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

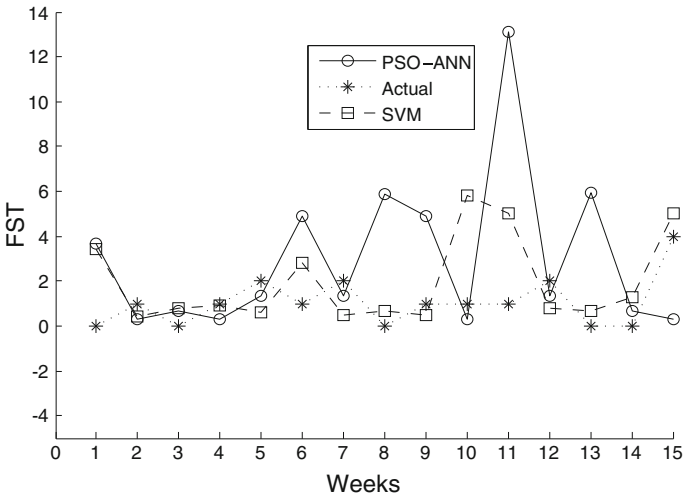


Fig. 11 Plot of the predicted versus the actual FST values at the function test phase for techniques having significant goodness of fit

The K–S test result for measuring the goodness of fit for predictions from each technique relative to the actual FST at the integration test phase appear in Table 6. Bagging, GP, AIRS, and human predictions show statistically significant goodness of fit. Figure 13 shows the line plots of Bagging, GP, AIRS, and the human predictions.

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

4.2.4 Prediction of FST at the system test phase

The box plots of absolute residuals for predicting FST at the system test phase for different techniques are shown in Fig. 14a. We can observe that there are certain techniques that appear to do better. These are PR, RF, and GP . The box plots of these three techniques have medians closer to the 0 mark on the *y*-axis, with GP being the closest. GP also show the smallest distribution as compared with PR and RF . For the rest of the techniques, there is a greater variance in their box plots with outliers. MR, Knn, AIRS, and human box plots

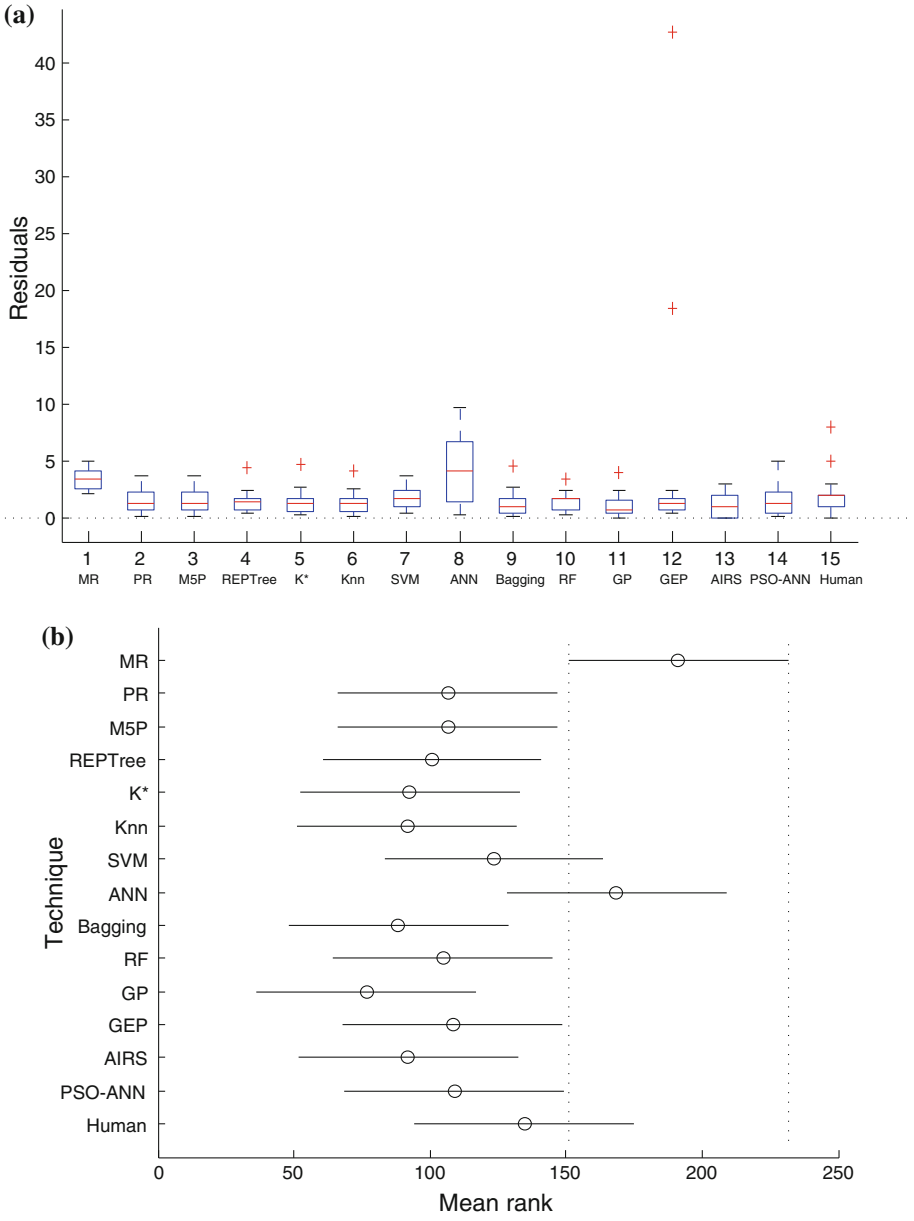


Fig. 12 Results showing *box plots* of absolute residuals and multiple comparisons of the absolute residuals between all techniques at the integration test phase. **a** *Box plots* of the residuals for each technique in predicting FST at the integration test phase. **b** Results of the multiple comparisons test with $\alpha = 0.05$ (the *vertical dotted lines* indicating that eleven techniques have mean ranks significantly different from MR)

seem to be worse, both in terms of the position of the median and the spread of the distribution. The result of the Kruskal–Wallis test ($p = 5.6e^{-7}$) at $\alpha = 0.05$ suggested that at least one sample median is significantly different from the others. Subsequently, the results of the multiple comparisons test (Tukey–Kramer, $\alpha = 0.05$) appear in Fig. 14b. The

Table 6 Two-sample two-sided K–S test results for predicting FST at the integration test phase with critical value $J_{0.05} = 0.5$

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

technique with smallest mean rank is GP, and there are no pairwise significant differences between GP and any of the techniques: PR, Bagging, RF, and PSO-ANN . This finding also confirms the trend from the box plots. MR is the worst performing technique and there are no pairwise significant differences between MR and any of the techniques: M5P, REPTree, K*, Knn, SVM, ANN, Bagging, GEP, AIRS, PSO-ANN, and Human.

The K–S test result for measuring the goodness of fit for predictions from each technique relative to the actual FST at the system test phase appear in Table 7. PR, GP,and PSO-ANN show statistically significant goodness of fit. Figure 15 shows the line plots of PR, GP, PSO-ANN with the actual FST at the system test phase.

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

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

The analysis done in the previous Sect. 4.2 would also allow us to answer the RQ.2 that questions whether other techniques better predict FST than human expert judgement. We now analyze the performance of human expert judgement versus other techniques for FST prediction at each of the four test phases.

4.3.1 Prediction of FST at the unit test phase

Figure 8b shows the results of the multiple comparisons test (Tukey–Kramer, $\alpha = 0.05$) for FST prediction at the unit test phase. Two techniques have their means significantly different (and worse) than the human expert judgement. These techniques are MR and ANN . Otherwise, there are no significant pairwise differences between the means of human expert judgement and rest of the techniques.

In terms of goodness of fit, Table 4 shows that AIRS and human expert judgement have statistically significant goodness of fit in comparison with other techniques.

4.3.2 Prediction of FST at the function test phase

Figure 10b shows the results of the multiple comparisons test (Tukey–Kramer, $\alpha = 0.05$) for FST prediction at the function test phase. Three techniques have their means significantly different (and better) than the human expert judgement. These techniques are PSO-ANN, SVM, and Knn. Otherwise, there are no significant pairwise differences between the means of human expert judgement and the rest of the techniques.

In terms of goodness of fit, Table 5 shows that human expert judgement has no significant goodness of fit in comparison with SVM and PSO-ANN.

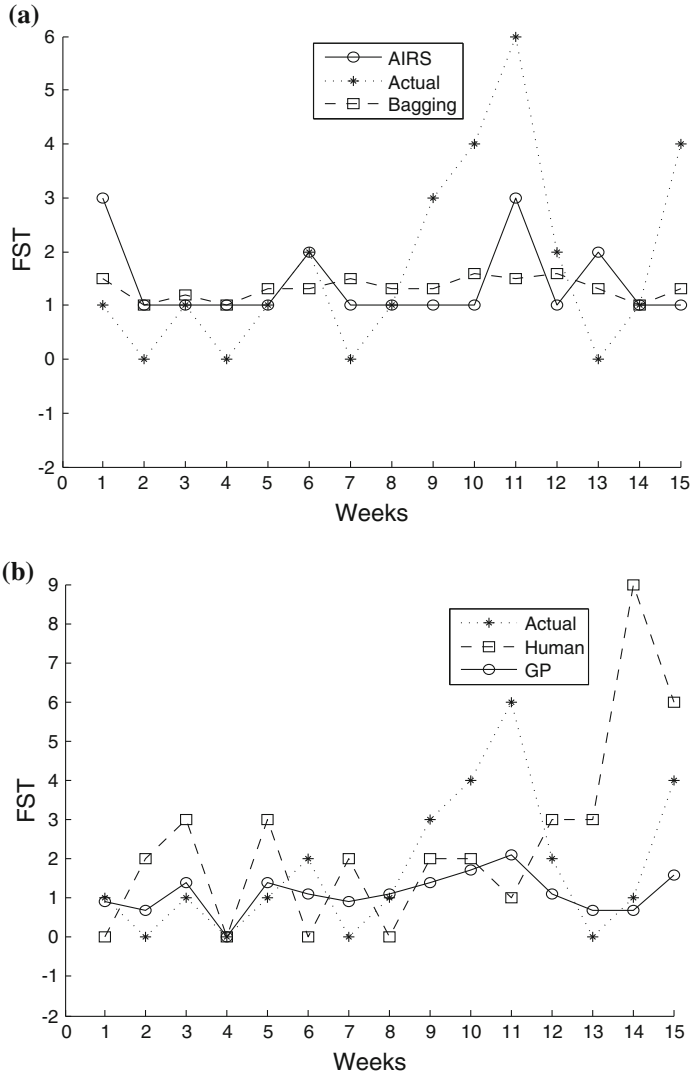


Fig. 13 Plot of the predicted versus the actual FST values at the integration test phase for techniques having significant goodness of fit. **a** Plot of the actual versus predicted FST values (AIRS and Bagging) at the integration test phase. **b** Plot of the actual versus predicted FST values (Human and GP) at the integration test phase

4.3.3 Prediction of FST at the integration test phase

Figure 12b shows the results of the multiple comparisons test (Tukey–Kramer, $\alpha = 0.05$) for FST prediction at the integration test phase. No technique has its mean significantly different than the human expert judgement.

In terms of goodness of fit, Table 6 shows that Bagging, GP, AIRS, and human expert judgement have significant goodness of fit in comparison with rest of the techniques.

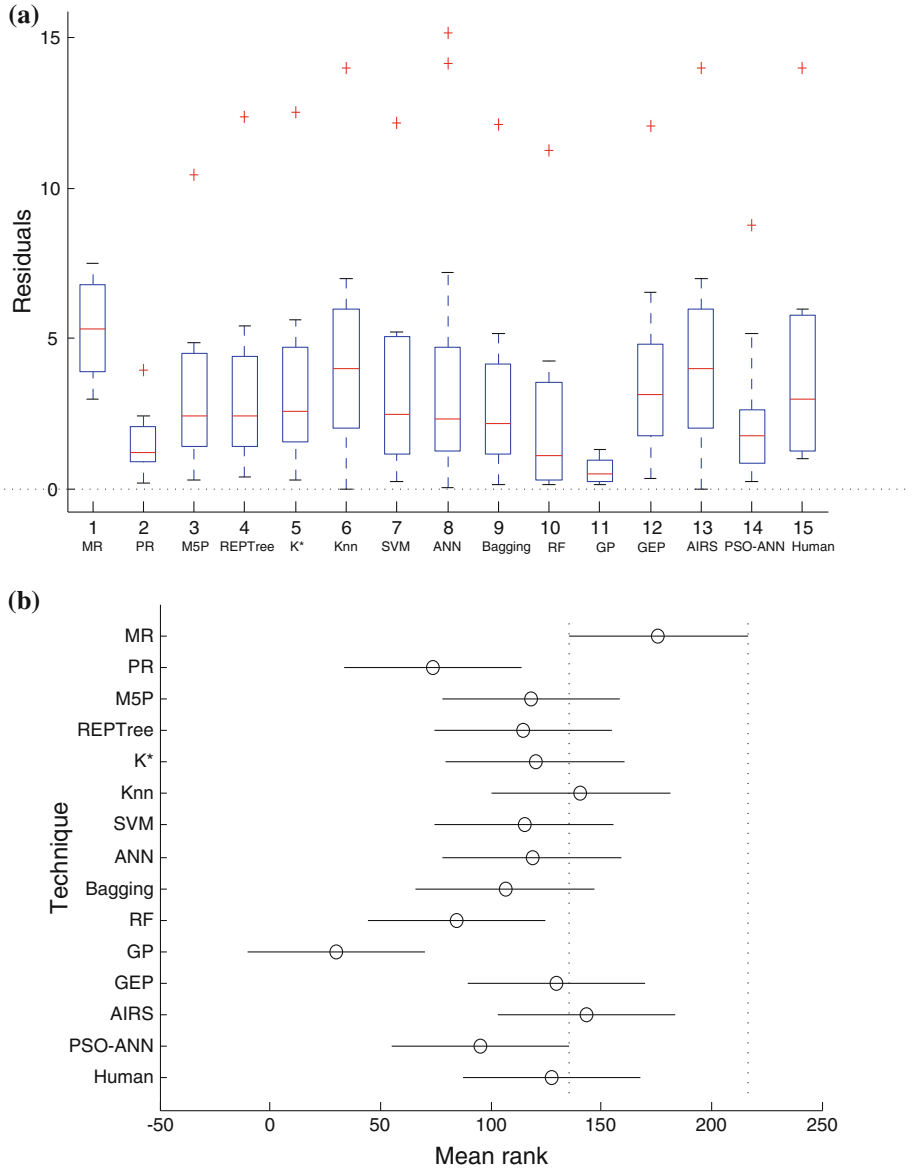


Fig. 14 Results showing *box plots* of absolute residuals and multiple comparisons of the absolute residuals between all techniques at the system test phase. **a** *Box plots* of the residuals for each technique in predicting FST at the system test phase. **b** Results of the multiple comparisons test with $\alpha = 0.05$ (the *vertical dotted lines* indicating that three techniques have mean ranks significantly different from MR)

4.3.4 Prediction of FST at the system test phase

Figure 14b shows the results of the multiple comparisons test (Tukey–Kramer, $\alpha = 0.05$) for FST prediction at the system test phase. GP has its mean significantly different (and better) than the human expert judgement.

Table 7 Two-sample two-sided K–S test results for predicting FST at the system test phase with critical value $J_{0.05} = 0.5$

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

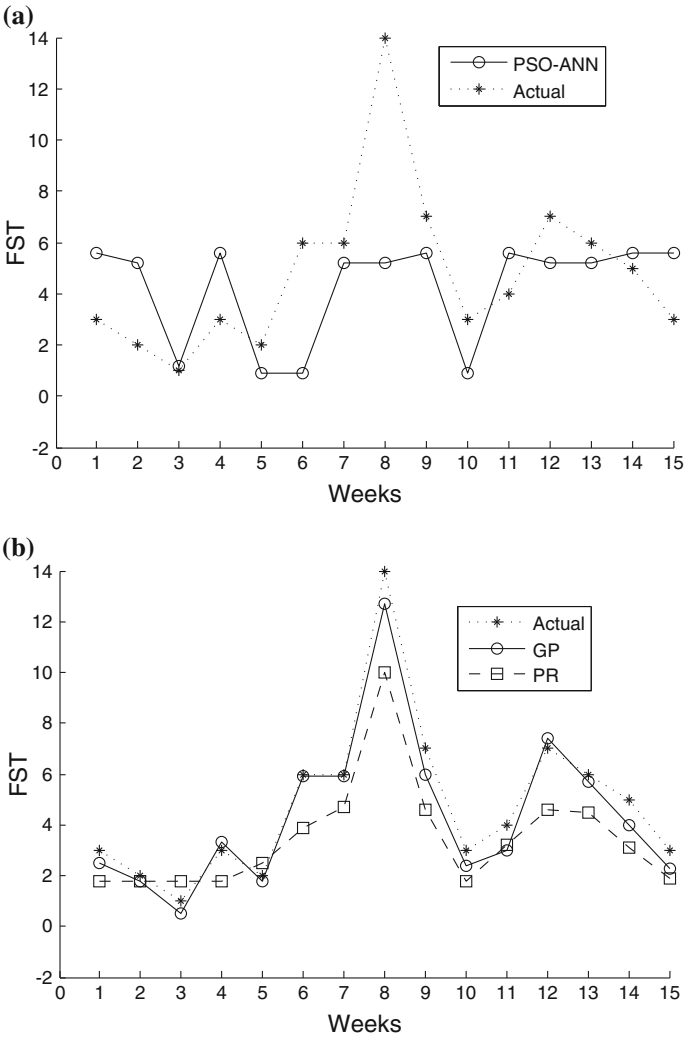


Fig. 15 Plot of the predicted versus the actual FST values at the system test phase for techniques having significant goodness of fit. **a** Plot of the actual versus predicted FST values (PSO-ANN) at the system test phase. **b** Plot of the actual versus predicted FST values (GP and PR) at the system test phase

Table 8 A summary of techniques that are or are not better than human experts predicting FST at unit, function, integration, and system test phases

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

In terms of goodness of fit, Table 7 shows that GP and PSO-ANN have significant goodness of fit in comparison with rest of the techniques.

Table 8 sums up which techniques are or are not better than human expert judgement in predicting FST at unit, function, integration,and system test phases. The dark gray cells in the Table 8 refer to techniques that are equally good in predicting FST with the human expert judgement. The light gray cells indicate that the techniques are inferior with respect to the human judgement and the dark gray cells. The white cells indicate that these techniques are better than human expert judgement in predicting FST.

5 Discussion

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

In this paper, we investigated two research questions outlined in Sect. 1. RQ.1 investigated the use of a variety of techniques for predicting FST in unit, function, integration, and system test phases. The results are evaluated for predictive accuracy (through absolute residuals) and goodness of fit (through the Kolmogorov-Smirnov test). A range of techniques are found to be useful in predicting FST for different test phases (both in terms of predictive accuracy and goodness of fit). RQ.2 is concerning a more specific research question that compared human expert judgement with other techniques. The results of this comparison indicate that expert human judgement is better than majority of the techniques at unit and integration test but are far off at function and system test. Hence, human predictions regarding FST lack some consistency. There are indications that a smaller group of techniques might be consistently better in predicting at all the test phases. Following is the list of techniques performing better at various test phases for predicting FST in our study:

1. Unit test—AIRS and human.
2. Function test—SVM and PSO-ANN.
3. Integration test—Bagging, GP, AIRS, human.
4. System test—PR, GP, PSO-ANN.

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

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

The search-based techniques have certain merits, one or more of which might be responsible for outperforming the other group of techniques:

- The search-based techniques are better able to cope with ill-defined, partial and messy input data (Harman 2010). GP is able to perform well where the interrelationships among the relevant variables are unknown or poorly understood (Poli et al. 2008). According to Poli et al. (2008), "[GP] has proved successful where the application is new or otherwise not well understood. It can help discover which variables and operations are important; provide novel solutions to individual problems; unveil unexpected relationships among variables; and sometimes GP can discover new concepts that can then be applied in a wide variety of circumstances." Evolutionary algorithms have also been applied successfully to problems where there are high correlations between variables, i.e., the choice of one variable may change the meaning or quality of another (Blickle 1996).
- GP is particularly good at providing small programs that are nearly correct and predictive models are not exceptionally long (Harman 2010). According to Poli et al. (2008),"[…]evolutionary algorithms tend to work best in domains where close approximations are both possible and acceptable." Search-based techniques can produce very transparent solutions, in the sense that they can make explicit the weight and contribution of each variable in the resulting solutions.
- Being nonparametric approaches, the structure of the end solution is not pre-conceived. This is particularly important for the usability of search-based techniques, i.e., the techniques used for prediction should be able to determine the form of relationship between inputs and outputs rather than that the technique is dependent on the user providing the form of the relationship.
- Search-based techniques are entirely data-driven approaches and do not include any assumptions about the distribution of the data in its formulation. For example, GP models are independent of any assumptions about the stochastic behavior of the software failure process and the nature of software faults (Afzal 2009).

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

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

Another important aspect of the results is that human predictions were among the better techniques for predicting FST at unit and integration test. In our view, this is also an important outcome and shows that expert opinions perhaps need more consideration that is largely been ignored in empirical studies of software fault predictions (Tomaszewski et al. 2007; Catal and Diri 2009). We, therefore, agree with the conclusion of Hughes (1996) that expert judgement should be supported by the use of other techniques rather than displacing it. Search-based techniques seem to be an ideal decision support tool for two reasons:

1. They have performed consistently better than other techniques (Sect. 4.2).

2. Search-based techniques, as part of the more general field of search-based software engineering (SBSE) (Harman and Jones 2001; Harman 2007), are inherently concerned with *improving* not with *proving* (Harman 2010).

As such, it is likely that *human-guided* semi-automated search might help get a reasonable solution that incorporates human judgement in the search process. This human-guided search is commonly referred to as 'human-in-the-loop' or 'interactive evolution' (Harman 2010) and is a promising area of future research. The incorporation of human feedback in the automated search can possibly account for some of the extreme fluctuations in the solely human predictions that are observed for predicting FST at unit and integration test.

We also believe that the selection of predictor variables that are easy to gather (e.g., the project level metrics at the subject company in this study) and that do not conflict with the development life cycle have better chances of industry acceptance. There is evidence to support that general process level metrics are more accurate than code/structural metrics (Arisholm et al. 2010). A recent study by Afzal (2010) has shown that the use of number of faults-slip-through to/from various test phases is able to provide good results for finding fault-prone modules at integration and system test phases. However, this subject requires further research.

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

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

6 Empirical validity evaluation

We adopted a case study approach in evaluating various techniques for predicting FST in four test phases. A controlled experiment was deemed not practical since too many human factors potentially affect fault occurrences.

What follows next is our presentation of the various threats to validity of our study: *Construct validity*. Our choice of selecting project level metrics (Table 1) instead of structural code metrics was influenced by multiple factors. First, metrics relevant to work packages (Sect. 3.1) have an intuitive appeal for the employees at the subject company where they can relate FST to the proportion of effort invested. Secondly, the existence of a module in multiple work packages made it difficult to obtain consistent metrics at the component level. Thirdly, the intent of this study is to use project level metrics that are readily available and hence reduces the cost of doing such predictions. In addition, the case study is performed in the same development organization having the identical application domain, so the two projects in focus are characterized by the same set of metrics. *Internal validity*. A potential threat to the internal validity is that the FST data did not consider the severity level of faults, rather treated all faults equally. As for the prediction techniques, the best we could do was to experiment with a variety of parameter values. But we acknowledge that the obtained results could be improved by better optimizing the parameters. *External validity*. The quantitative data modeling was performed on data from

a specific company, while the questionnaire was filled out by an expert having 20 years of work experience and currently holds the designation of systems verification process leader at our subject organization. The questionnaire was filled to provide expert estimations of FST metric and consisted of all relevant independent variables. The expert then used these independent variables to provide estimated values. In order to reduce bias arising from the design of the questionnaire, the researchers encouraged the expert for asking questions to clarify any ambiguities. We have tried to present the context and the processes to the extent possible for fellow researchers to generalize our results. We are also encouraged by the fact that the companies are enterprise-size and have development centers worldwide that follow similar practices. It is therefore likely that the results of this study are useful for them too. A threat to the external validity is that we cannot publicize our industrial data sets due to proprietary concerns. However, the transformed representation of the data can be made available if requested. *Conclusion validity*. We were conscious in using the right statistical test, basing our selection on whether the assumptions of the test were met or not. We used a significance level of 0.05, which is a commonly used significance level for hypothesis testing (Juristo and Moreno 2001), however, facing some criticism lately (Ioannidis 2005).

7 Conclusion

In this paper, we have presented an extensive empirical evaluation of various techniques for predicting the number of faults slipping through to the four test phases of unit, function, integration, and system.

We find that a range of techniques are found to be useful in such a prediction task, both in terms of predictive accuracy and goodness of fit. However, the group of search-based techniques [genetic programming (GP), gene expression programming (GEP), artificial immune recognition system (AIRS), and particle swarm optimization–based artificial neural network (PSO-ANN)] consistently give better predictions, having a representation at all of the test phases. Human predictions are also among the better techniques at two of the four test phases. We conclude that human predictions can be supported well by the use of search-based techniques and a mix of the two approaches has the potential to provide improved results.

It is important to highlight that there might be additional evaluation criteria that are important in addition to measuring the predictive accuracy and the goodness of fit. A general multi-criteria based evaluation system is then required that captures both the quantitative and the qualitative aspects of such a prediction task. Future work will also investigate ways to incorporate human judgement in the automated search mechanism.

There are some lessons learnt at the end of this study which might be useful for decision-making in real-world industrial projects:

- The number of faults slipping through to various test phases can be decreased if improvement measures are taken in advance. To achieve this, prediction of FST is an important decision-making tool.
- In large industrial projects, prediction of FST is possible using project level measurements such as fault-inflow, status ranking of work packages, and test case progress.
- The organization wanting to improve their software testing process needs to institutionalize a mechanism for recording data in a consistently correct manner.
- Tool support that uses the recorded data and applies a number of techniques to provide results to the software engineer will improve usability of any prediction effort, including FST prediction.

- Tool support is also necessary to hide complex implementation details of techniques and to ease parameter settings for end users.
- The software engineers need to be trained in the prediction task. Training workshops need to be conducted which will not only increase awareness about the potential benefits of FST prediction but will also help to discover potentially new independent variables of interest.

Acknowledgment We are grateful to Prof. Anneliese Andrews, University of Denver, for reading and commenting on the initial concept paper.

Appendix: Model training and testing procedure

This section discusses the parameter settings that have been considered for different techniques during model selection. These settings may be used for a future replication of this study and to quantify the impact of changing the parameter settings, perhaps using different data sets. As given in Sect. 3.1, we use data from 45 weeks of the baseline project to train the models, while the results are evaluated on the data from 15 weeks of an ongoing project. The experimental evaluation process is also summarized in Procedure 1.

Procedure 1 The model training and testing procedure.

```
Train: Training dataset of 45 weeks
Test: Testing dataset of 15 weeks
P: Set of parameter settings for each technique
T: Set of techniques
for each t in T do
  for each p in P do
    Model = BuildModel (Train, t, P[t])
    ARE [t, p] = GetResult (Test, Model)
    output ARE
  end for
  return parameter[min(ARE)]
end for
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
BuildModel (TrainingData, Technique, ParameterSet)
// Train the technique using training data with parameters
GetResult (TestData, Model)
// Compute ARE of the trained model on the Test data
```

The least-square multiple regression does not require selection of parameters, rather the coefficients are determined from the training data. Different estimators implemented in the WEKA machine-learning tool (Hall et al. 2009) have been evaluated for pace regression that includes empirical Bayes, ordinary least square, Akaike's information criterion (AIC), and risk inflation criterion (RIC). The estimator giving the least ARE is selected as the best pace regression model.

The M5P technique requires setting the minimum number of instances at a leaf node and has been varied in the range [2, 4, …, 10] with pruning and smoothing. The model with minimum ARE is retained. The REPTree technique requires setting the maximum depth of the tree, the minimum total weight of the instances in a leaf, the minimum variance proportion at a node required for splitting, the number of folds of data used for pruning, and the seed value used for randomizing the data. We have imposed no restriction on the

maximum depth of the tree while the minimum total weight of the instances in a leaf is varied in the range [2,4, …, 10]. The minimum variance proportion at a node, the number of folds of data used for pruning, and the seed value used for randomization are kept constant at their default values of 0.0010, 3, and 1, respectively.

The K* instance-based technique requires setting the blending parameter that has a value between 0 and 100 %. This parameter has been varied in the range of [0, 20, 40, …, 100]. For k-NN, the number of neighbors has been varied in the range of [1, 3, 5, …, 15].

For SVM, two types of parameters have to be set by the user, i.e., values for the epsilon parameter, ε and the regularization parameter, C . Setting the value of C near the range of the output values has been found to be a successful heuristic. We therefore vary C within the range [1, 3, …, 11]. The value of ε is varied in the range [0.001, 0.003] while the kernel used is the radial basis function. Training an artificial neural network (ANN) requires deciding on the number of layers and the number of nodes at each layer. We considered the ANN architecture with 1 input layer, 2 hidden layers, and 1 output layer. The number of independent variables in the problem determined the number of input nodes. The two hidden layers used a varied number of nodes in the range [1, 3, 5, 7], while the output layer used a single node. The hyperbolic tangent sigmoid and linear transfer functions have been used for the hidden and output nodes, respectively. Finally, the number of epochs used is 500 and the weights are updated using a learning rate of 0.3 and a momentum of 0.2.

Model selection for Bagging involves deciding upon the size of the bag as a percentage of the training set size and the number of iterations to be performed. These two parameters have been varied in the range [25, 50, 75, 100] and [5, 10, 15], respectively. The REPTree technique is used as the base learner. For rotation forest, the number of iterations have been varied in the range [5, 10, 15] and the base learner used is the REPTree technique.

GP requires setting a number of control parameters. Although the affect of changing these control parameters on the end solution is still an active area of research, we nevertheless experimented with different function and terminal sets. Initially, we experimented with a minimal set of functions and the terminal set containing the independent variable only. We incrementally increased the function set with additional functions and later on also complemented the terminal set with a random constant. The best model having the best fitness was chosen from all the runs of the GP system with different variations of function and terminal sets. The GP programs were evaluated according to the sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n} |e_i - e_i'|$, 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. The control parameters that were chosen for the GP system are shown in Table 9. For GEP, the

Table 9 GP control parameters

Control parameter	Value
Population size	50
Termination condition	2,000 generations
Function set	{ +, −, *, /, 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

Table 10 GEP control parameters

Control parameter	Value
Population size	50
Genes per chromosome	4
Gene head length	8
Termination condition	2,000 generations
Functions	{ +, −, *, /, 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

solutions are evaluated for fitness using mean squared error and the control parameters are shown in Table 10. The AIRS algorithm also requires setting a number of parameters. While it is not possible to experiment with all the different combinations of these parameters, however, the value of k for the majority voting has been varied in the range [1, 3, 5, …, 15]. Rest of the parameters used were as follows: affinity threshold = 0.2, clonal rate = 10, hypermutation rate = 2, mutation rate = 0.1, stimulation value = 0.9, and total resources = 150. For PSO-ANN, the architecture similar to the basic ANN is followed except that the weights are now optimized using PSO with the number of particles in the swarm set to 25 and the number of iterations varied in the range [500, 1,000, 15,000, 2,000]. The mean squared error is used as the fitness function.

References

Afzal, W. (2009). *Search-based approaches to software fault prediction and software testing*. Blekinge Institute of Technology Licentiate Series No. 2009:06, Ronneby, Sweden.

Afzal, W. (2010). *Using faults-slip-through metric as a predictor of fault-proneness: Proceedings of the 21st Asia Pacific Software Engineering Conference (APSEC'10), IEEE.*

Afzal, W., & Torkar, R. (2008). *A comparative evaluation of using genetic programming for predicting fault count data: Proceedings of the 3rd International Conference on Software Engineering Advances (ICSEA'08), IEEE.*

Afzal, W., Torkar, R., Feldt, R., & Wikstrand, G. (2010). *Search-based prediction of fault-slip-through in large software projects: Proceedings of the 2nd International Symposium on Search-Based Software Engineering (SSBSE'10), IEEE Computer Society*. pp. 79–88.

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*, 37–66.

Antolić, Ž. (2007). *Fault slip through measurement process implementation in CPP software implementation: Proceedings of the 30th Jubilee International Convention (MIPRO'07)*. Ericsson Nikola Tesla.

Arisholm, E., Briand, L. C., & Johannessen, E. B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software, 83*(1), 2–17.

Blickle, T. (1996). Theory of evolutionary algorithms and application to system synthesis. PhD thesis. Zurich, Switzerland: Swiss Federal Institute of Technology.

Boehm, B., & Basili, V.R. (2001). Software defect reduction top 10 list. *Computer, 34*(1), 135–137.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*, 123–140.

Briand, L., Emam, K., Freimut, B., & Laitenberger, O. (2000). A comprehensive evaluation of capture-recapture models for estimating software defect content. *IEEE Transactions on Software Engineering, 26*(6).

Canu S, Grandvalet Y, Guigue V, & Rakotomamonjy A (2005) SVM and kernel methods toolbox. Perception Systémes et Information, INSA de Rouen, Rouen, France.

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

Challagulla, V., Bastani, F., Yen, I., & Paul, R. (2005). Empirical assessment of machine learning based software defect prediction techniques. *Proceedings of the 10th IEEE workshop on object oriented real-time dependable systems*.

Cleary, J. G., & Trigg, L. E. (1995). K*: An instance-based learner using an entropic distance measure. *12th International Conference on Machine Learning (ICML'95)*.

Damm, L. O. (2007). *Early and cost-effective software fault detection—Measurement and implementation in an industrial setting*. PhD thesis, Blekinge Institute of Technology.

Damm, L. O., Lundberg, L., & Wohlin, C. (2006). Faults-slip-through—a concept for measuring the efficiency of the test process. *Software Process: Improvement & Practice, 11*(1), 47–59.

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

Ferreira, C. (2001). Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems, 13*(2).

Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering, 31*(10). 897–910.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009), The WEKA data mining software: An update. *SIGKDD Explorations Newsletter, 11*(1), 10–18.

Harman, M. (2007). The current state and future of search based software engineering. *Proceeding of the Future of Software Engineering (FOSE'07)*. Washington, DC, USA: IEEE Computer Society, pp. 342–357.

Harman, M. (2010). *The relationship between search based software engineering and predictive modeling: Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE'10)*. New York, NY, USA: ACM.

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

Hribar, L. (2008) *Implementation of FST in design phase of the project: Proceedings of the 31st Jubilee International Convention (MIPRO'08)*. Ericsson Nikola Tesla.

Hughes, R. T. (1996). Expert judgement as an estimating method. *Information and Software Technology, 38*(2), 67–75.

IEEE. (1990) *IEEE standard glossary of software engineering terminology—IEEE Std 610.12-1990. Standards Coordinating Committee of the Computer Society of the IEEE, IEEE Standards Board*. New York, USA: The Institute of Electrical and Electronic Engineers, Inc.

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

Jha, G. K., Thulasiraman, P., & Thulasiram, R. K. (2009). *PSO based neural network for time series forecasting: International Joint Conference on Neural Networks*.

Jørgensen, M., Kirkebøen, G., Sjøberg, D. I. K., Anda, B., & Bratthall, L. (2000). *Human judgement in effort estimation of software projects: Proceedings of the workshop on using multi-disciplinary approaches in empirical software engineering research, co-located with ICSE'00*. Ireland: Limerick.

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

Kachigan, S. K. (1982). *Statistical analysis—an interdisciplinary introduction to univariate and multivariate methods*. New York: Radius Press.

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

Lavesson, N., & Davidsson, P. (2008). *Generic methods for multi-criteria evaluation: Proceedings of the SIAM International Conference on Data Mining (SD'08)*.

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering, 34*(4), 485–496.

Liu, Y., Khoshgoftaar, T., & Seliya, N. (2010). Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Transactions on Software Engineering* (Article in print).

Lyu, M.R. (ed) (1996). *Handbook of software reliability engineering*. Hightstown, NJ: McGraw-Hill Inc.

Mohagheghi, P., Conradi, R., Killi, O. M., & Schwarz, H. (2004). *An empirical study of software reuse vs. defect-density and stability: Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*. Washington, DC, USA: IEEE Computer Society.

Nagappan, N., & Ball, T. (2005). *Static analysis tools as early indicators of pre-release defect density: Proceedings of the 27th international conference on Software engineering (ICSE'05)*. New York, NY, USA: ACM.

Nagappan, N., Murphy, B., & Basili, V. (2008). *The influence of organizational structure on software quality: An empirical case study: Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*. New York, NY, USA: ACM.

Pickard, L., Kitchenham, B., & Linkman, S. (1999). An investigation of analysis techniques for software datasets. In: *Proceedings of the 6th International Software Metrics Symposium (METRICS'99)*. Los Alamitos, USA: IEEE Computer Society.

Poli, R., Langdon, W. B., & McPhee, N. F. (2008). A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk.

Rakitin, S. R. (2001). *Software verification and validation for practitioners and managers* (2nd ed.). 685 Canton Street, Norwood, MA, USA: Artech House., Inc.

Rodriguez, J. J., Kuncheva, L. I., & Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 28*, 1619–1630.

Runeson, P., Andersson, C., Thelin, T., Andrews, A., & Berling, T. (2006). What do we know about defect detection methods? *IEEE Software, 23*(3), 82–90.

Russell S., & Norvig P. (2003) Artificial intelligence—a modern approach. Prentice Hall Series in Artificial Intelligence, USA

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

STD. (2008) *IEEE standard 12207-2008 systems and software engineering—Software life cycle processes. Software and systems engineering standards committee of the IEEE computer society*. New York, USA: The Institute of Electrical and Electronic Engineers, Inc.

Staron, M., & Meding, W. (2008). Predicting weekly defect inflow in large software projects based on project planning and test status. *Information & Software Technology, 50*(7–8), 782–796.

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

Tomaszewski, P., Håkansson, J., Grahn, H., & Lundberg, L. (2007). Statistical models vs. expert estimation for fault prediction in modified code—an industrial case study. *Journal of Systems and Software, 80*(8), 1227–1238.

Trelea, I. C. (2003). The PSO algorithm: Convergence analysis and parameter selection. *IP Letters, 85*(6).

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

Wagner, S. (2006). *A literature survey of the quality economics of defect-detection techniques: Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering (ISESE'06)*

Wang, Y. (2000). *A new approach to fitting linear models in high dimensional spaces. PhD thesis*. New Zealand: Department of Computer Science, University of Waikato.

Wang, Y, & Witten, I. H. (1996). *Induction of model trees for predicting continuous classes*. Technical report, University of Waikato, Department of Computer Science, Hamilton, New Zealand, URL http://www.cs.waikato.ac.nz/pubs/wp/1996/uow-cs-wp-1996-23.pdf.

Watkins, A., Timmis, J., & Boggess, L. (2004). Artificial immune recognition system (AIRS): An immune-inspired supervised learning algorithm. *Genetic programming and Evolvable Machines, 5*(3), 291–317.

Weka Documentation (2010) Class REPTree.Tree. URL http://www.dbs.ifi.lmu.de/~zimek/diplomathesis/implementations/EHNDs/doc/weka/classifiers/trees/REPTree.Tree.html.

Weyuker, E. J., Ostrand, T. J., & Bell, R. M. (2010). Comparing the effectiveness of several modeling methods for fault prediction. *Empirical Software Engineering, 15*(3), 277–295.

Witten I, Frank E (2005) Data mining—practical machine learning tools and techniques. USA: Morgan–Kaufmann Publishers

Zhong, S., Khoshgoftaar, T. M., & Seliya, N. (2004). *Unsupervised learning for expert-based software quality estimation: Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering (HASE'04)*.

Author Biographies

Dr. Wasif Afzal is an Assistant Professor of Software Engineering at Bahria University, Islamabad, Pakistan. Before, he was a postdoctoral researcher and a PhD student at Blekinge Institute of Technology, Sweden. His current research focuses on empirical evaluation of different techniques for software fault prediction. In particular, he is interested in evolutionary computation approaches for software fault prediction. He has over two years of industrial experience in software quality and software testing. He is a member of IEEE.

Dr. Richard Torkar received his BSc in Computer Science from University West, Sweden, and later the PhD degree in Software Engineering from Blekinge Institute of Technology, Sweden. He is currently an Associate Professor of Software Engineering at Chalmers Institute of Technology.He performs empirical research in software engineering in general, and software testing and software reliability in particular. His research focuses on applying different combinations of search-based strategies to software engineering activities.

Dr. Robert Feldt is an Associate Professor of Software Engineering at Chalmers Institute of Technology in Sweden. He received a M.Sc. degree from Chalmers University of Technology in 1997 and a PhD degree from the same university in 2002. His main research interestsinclude software testing and verification and validation, automated and biomimetic software engineering, psychology of programming, user experience and human-centered software engineering. The research is conducted mainly through empirical methods such as controlledexperiments, surveys and case studies, but also through technical and theory development. In parallel with his academic projects, he helps software developing organizations and startups. Dr. Feldt has published more than 30 papers in international journals, conference and workshop proceedings. He is a member of IEEE.

Dr. Tony Gorschek (please see http://www.gorschek.com) is an Associate Professor of Software Engineering at Blekinge Institute of Technology (BTH). He holds a PhD in Software Engineering and a Bachelor in Economics from BTH. Prior to, and in parallel with, his academic career, Dr. Gorschek has worked as a consultant in industry and has also held the positions of CTO, chief architect, and project manager in a number of companies doing development of software intensive systems, totaling over ten years industrial experience. At present, Dr. Gorschek manages his own consultancy company and holds the position of CTO in Qjuub AB, inparallel with a full-time research position. His research interests include requirements engineering, technical product management, process assessment and improvement, quality assurance, and innovation. Most of the research is conducted in close collaboration andcooperation with industry partners such as ABB and Ericsson. He is a member of the IEEE and the ACM.