Expert Systems with Applications 38 (2011) 11984–11997

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Review

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

Wasif Afzal [*], Richard Torkar

Blekinge Institute of Technology, S-371 79 Karlskrona, Sweden

ARTICLE INFO

Keywords:
Systematic review
Genetic programming
Symbolic regression
Modeling

ABSTRACT

The objective of this paper is to investigate the evidence for symbolic regression using genetic programming (GP) being an effective method for prediction and estimation in software engineering, when compared with regression/machine learning models and other comparison groups (including comparisons with different improvements over the standard GP algorithm). We performed a systematic review of literature that compared genetic programming models with comparative techniques based on different independent project variables. A total of 23 primary studies were obtained after searching different information sources in the time span 1995–2008. The results of the review show that symbolic regression using genetic programming has been applied in three domains within software engineering predictive modeling: (i) Software quality classification (eight primary studies). (ii) Software cost/effort/size estimation (seven primary studies). (iii) Software fault prediction/software reliability growth modeling (eight primary studies). While there is evidence in support of using genetic programming for software quality classification, software fault prediction and software reliability growth modeling; the results are inconclusive for software cost/effort/size estimation.

1. Introduction

Evolutionary algorithms represent a subset of the metaheuristic approaches inspired by evolution in nature, (Burke & Kendall, 2005) such as reproduction, mutation, cross-over, natural selection and survival of the fittest. All evolutionary algorithms share a set of common properties (Bäck, Fogel, & Michalewicz, 2000):

1. These algorithms work with a population of solutions, utilizing a collective learning process. This population of solutions make-up the search space for the evolutionary algorithms.
2. The solutions are evaluated by means of a quality or fitness value whereby the selection process promotes better solutions than those that are worse.
3. New solutions are generated by random variation operators intended to model mutation and recombination.

The above process is iterated over successive generations of evaluation, selection and variation, with a hope that increasingly suitable solutions would be found, i.e., the search is guided to more feasible locations of the search space. Examples of typical evolutionary algorithms include genetic algorithms, evolution strategies and evolutionary programming. The structure of an evolutionary algorithm is given below (Michalewicz & Fogel, 2004):

```
procedure evolutionary algorithm
P(t): population of solutions; t: iteration number;
begin
    t ← 0
    initialize P(t)
    evaluate P(t)
    while (not termination-condition) do
    begin
        t ← t + 1
        select P(t) from P(t − 1)
        alter P(t)
        evaluate P(t)
    end
end
```

The evolutionary algorithm maintains a population of solutions $P(t)$ for iteration t. Each solution in the population is evaluated for its fitness (the "evaluate step"). A new population at iteration $t + 1$ is formed by selecting the more fitter solutions (the "select step"). Some solutions of the new population undergo transformations (the "alter step") using the variation operators. The algorithm is iterated until a predefined termination criterion is reached.

* Corresponding author. Tel.: +46 455 385840; fax: +46 455 385057.
E-mail addresses: wasif.afzal@bth.se (W. Afzal), richard.torkar@bth.se (R. Torkar).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.03.041

Genetic programming (GP) (Banzhaf, Nordin, Keller, & Francone, 1998; Koza, 1992; Poli, Langdon, & McPhee, 2008) is an extension of genetic algorithms. It is a *systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done* (Poli et al., 2008). Like other evolutionary algorithms, GP applies iterative, random variation to an existing pool of computer programs (potential solutions to the problem) to form a new generation of programs by applying analogs of naturally occurring genetic operations (Koza & Poli, 2005). Programs may be expressed in GP as syntax trees, with the nodes indicating the instructions to execute (called functions), while the tree leaves are called terminals and may consist of independent variables of the problem and random constants. To use GP one usually needs to take five preparatory steps (Koza & Poli, 2005):

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

The first two steps define the search space that will be explored by GP. The fitness measure guides the search in promising areas of the search space and is a way of communicating a problem's requirements to a GP algorithm. The fitness evaluation of a particular individual is determined by the correctness of the output produced for all of the fitness cases (Bäck et al., 2000). The last two steps are administrative in their nature. The control parameters limit and control how the search is performed like setting the population size and probabilities of performing the genetic operations, while the termination criterion specifies the ending condition for the GP run and typically includes a maximum number of generations (Koza & Poli, 2005). Genetic operators of mutation, crossover and reproduction are mainly responsible for introducing variation in successive generations. The crossover operator recombines randomly chosen parts from two selected programs and creates new program(s) for the new population. The mutation operator selects a point in a parent tree and generates a new random sub-tree to replace the selected subtree, while the reproduction operator simply replicates a selected individual to a new population.

Symbolic regression is one of the many application areas of GP which finds a function, with the outputs having desired outcomes. It has the advantage of being independent of making any assumptions about the function structure. Another potential advantage is that models built using symbolic regression application of GP can also help in identifying the significant variables which might be used in subsequent modeling attempts (Kotanchek, Smits, & Kordon, 2003).

This paper reviews the available literature on the application of symbolic regression using GP for predictions and estimations within software engineering. The performance of symbolic regression using GP is assessed in terms of its comparison with competing models which might include common machine learning models, statistical models, models based on expert opinion and models generated using different improvements over the standard GP algorithm. There are two reasons for carrying out this study:

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

The authors are not aware of any study having goals similar to ours. Prediction and estimation in software engineering has been applied to measure different attributes. A non-exhaustive list includes prediction and estimation of software quality, e.g. Lanubile and Visaggio (1997), software size, e.g. Low and Jeffery (1990), software development cost/effort, e.g. Jørgensen and Shepperd (2007), maintenance task effort, e.g. Jørgensen (1995), correction cost, e.g. de Almeida, Lounis, and Melo (1998), software fault, e.g. Thelin (2004), and software release timing, e.g. Dohi, Nishio, and Osaki (1999). A bulk of the literature contributes to software cost/effort and software fault prediction. A systematic review of software fault prediction studies is given by Catal and Diri (2009), while a systematic review of software development cost estimation studies is provided by Jørgensen and Shepperd (2007). The current study differs from these systematic reviews in several ways. Firstly, the studies as in Catal and Diri (2009) and Jørgensen and Shepperd (2007) are more concerned with classification of primary studies and capturing different trends. This is not the primary purpose of this study which is more concerned with investigating the comparative efficacy of using symbolic regression across software engineering predictive studies. Secondly, Catal and Diri (2009) and Jørgensen and Shepperd (2007) review the subject area irrespective of the applied method, resulting in being more broad in their coverage of the specific area. This is not the case with this study as it is narrowly focused in terms of the applied technique and open in terms of capturing prediction and estimation of different attributes (as will be evident from the addressed research question in Section 2.1). Thirdly, one additional concern, which makes this study different from studies of Catal and Diri (2009) and Jørgensen and Shepperd (2007), is that it also assesses the evidence of comparative analysis of applying symbolic regression using GP with other competing techniques.

A paper by Crespo, Cuadrado, Garcia, Marban, and Sanchez-Segura (2003) presents a classification of software development effort estimation into artificial intelligence (AI) methods of neural networks, case-based reasoning, regression trees, fuzzy logic, dynamical agents and genetic programming. While the authors were able to present a classification scheme, it is not complete in terms of its coverage of studies within each AI method.

One other motivation of us carrying out this systematic review is the general growing interest in search-based approaches to solve software engineering problems (Harman & Jones, 2001). In this regards, it is interesting to investigate the extent of application of genetic programming (a search-technique) within software engineering predictive modeling. This presents an opportunity to assess different attributes which can be measured using GP. It also allows us to gain an understanding of different improvements/variations used by these studies to predict and estimate in a better way.

In rest of the text below, wherever we refer to GP, we mean the symbolic regression application of it.

This paper is organized as follows: Section 2 describes the research method including the research question, the search strategy, the study selection procedure, the study quality assessment and the data extraction. Results are presented in Section 3, while Section 4 discusses the results and future work. Validity threats and conclusions appear in Sections 5 and 6, respectively.

2. Method

This section describes our review protocol, which is a multistep process following the guidelines outlined in Kitchenham (2007).

2.1. Research question

We formulated the following research question for this study:

RQ Is there evidence that symbolic regression using genetic programming is an effective method for prediction and estimation, in comparison with regression, machine learning and other models (including expert opinion and different improvements over the standard GP algorithm)?

The research questions can conveniently be structured in the form of PICOC (Population, Intervention, Comparison, Outcome, Context) criteria (Petticrew & Roberts, 2005). The *population* is this study is the domain of software projects. *Intervention* includes models evolved using symbolic regression application of GP. The *comparison* intervention includes the models built using regression, machine learning and other methods. The *outcome* of our interest represents the comparative effectiveness of prediction/estimation using symbolic regression and machine learning/regression/other models (including different improvements over the standard GP algorithm). We do not pose any restrictions in terms of context and experimental design.

2.2. The search strategy

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

1. *Breaking down the research question into PICOC criteria.* This is done to manage the complexity of a search string that can get sophisticated in pursuit of comprehensiveness.
2. *Identification of alternate words and synonyms for each of PICOC criterion.* First, since it is common that terminologies differ in referring to the same concept, derivation of alternate words and synonyms helps ensuring completeness of search. The genetic programming bibliography maintained by Langdon, Gustafson, and Koza (2009) and Alander's bibliography of genetic programming (Alander, 2009) turned out to be valuable sources for deriving the alternate words and synonyms. Secondly our experience of conducting studies in a similar domain was also helpful (Afzal, Torkar, & Feldt, 2009).
3. *Use of Boolean* `OR` *to join alternate words and synonyms.*
4. *Use of Boolean* `AND` *to join major terms.*

We came up with the following search terms (divided according to the PICOC criteria given in Section 2.1):

- **Population.** Software, application, product, web, Internet, World-Wide Web, project, development.
- **Intervention.** Symbolic regression, genetic programming.
- **Comparison intervention.** Regression, machine learning, machine-learning, model, modeling, modelling, system identification, time series, time-series.
- **Outcomes.** Prediction, assessment, estimation, forecasting.

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

The search string was applied to the following digital libraries, while searching within all the available fields:

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

In order to ensure the completeness of the search strategy, we compared the results with a small core set of primary studies we found relevant, i.e. (Burgess & Lefley, 2001; Costa, de Souza, Pozo, & Vergilio, 2007; Dolado, 2001). All of the known papers were found using multiple digital libraries.

We additionally scanned the online GP bibliography maintained by Langdon et al. (2009) by using the search-term *symbolic regression*. We also searched an online data base of software cost and effort estimation called BESTweb (Jørgensen, 2009), using the search-term *genetic programming*.

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

The time span of the search had a range of 1995–2008. The selection of 1995 as the starting year was motivated by the fact that we did not find any relevant study prior to 1995 from our search of relevant GP bibliographies (Alander, 2009; Langdon et al., 2009). In addition, we also did not find any relevant study published before 1995 as a result of scanning of the reference lists of studies found by searching the electronic databases.

2.3. The study selection procedure

The purpose of the study selection procedure is to identify primary studies that are directly related to answering the research question (Kitchenham, 2007). We excluded studies that:

1. Do not relate to software engineering or software development, e.g. Alfaro-Cid, McGookin, Murray-Smith, and Fossen (2008).
2. Do not relate to prediction/estimation of software cost/effort/size, faults, quality, maintenance, correction cost and release timing, e.g. Abraham (2008).
3. Report performance of a particular technique/algorithmic improvement without being applied to software engineering, e.g. Andersson, Svensson, Nordin, and Nordahl (1999).
4. Do not relate to symbolic regression (or any of its variants) using genetic programming, e.g. Shukla (2000).
5. Do not include a comparison group, e.g. Khoshgoftaar and Liu (2007).
6. Do not include a valid comparison group, e.g. in Evett, Khoshgoftar, der Chien, and Allen (1998) where GP is compared with random rank ordering of software modules which is not a valid technique for software quality classification.
7. Use genetic programming only for feature selection prior to using some other technique, e.g. Ratzinger, Gall, and Pinzger, 2007.

Table 1
Count of papers before and after duplicate removal for the digital search in different publication sources. The numbers within parenthesis indicates the counts after duplicate removal.

Source	Count
EI Compendex & Inspec	578 (390)
ScienceDirect	496 (494)
IEEE Xplore	55 (55)
ISI Web of Science	176 (176)
ACM Digital Library	1081 (1081)
Langdon et al. GP bibliography (Langdon et al., 2009)	342 (342)
BESTweb (Jørgensen, 2009)	4 (4)
Total	2732 (2542)

8. Represent similar studies, i.e., when a conference paper precedes a journal paper. As an example, we include the journal article by Costa et al. (2007) but exclude two of their conference papers (Costa, Pozo, & Vergilio, 2006; Costa, Vergilio, Pozo, & de Souza, 2005).

Table 1 presents the count of papers and the distribution before and after duplicate removal as a result of the automatic search in the digital libraries.

The exclusion was done using a multi-step approach. First, references were excluded based on title and abstract which were clearly not relevant to our research question. The remaining references were subject to a detailed exclusion criteria (see above) and, finally, consensus was reached among the authors in including 23 references as primary studies for this review.

2.4. Study quality assessment and data extraction

The study quality assessment can be used to devise a detailed inclusion/exclusion criteria and/or to assist data analysis and synthesis (Kitchenham, 2007). We did not rank the studies according to an overall quality score but used a simple 'yes' or 'no' scale (Dybå, Dingsøyr, & Hanssen, 2007). Table 10, in Appendix A, shows the application of the study quality assessment criteria where a ($\surd$) indicates 'yes' and ($\times$) indicates 'no'. Further a ($\sim\surd$) shows that we were not sure as not enough information was provided but our inclination is towards 'yes' based on reading full text. A ($\sim\times$) shows that we were not sure as not enough information was provided but our inclination is towards 'no' based on reading full text. We developed the following study quality assessment criteria, taking guidelines from Kitchenham (2007) and Kitchenham, Mendes, and Travassos (2007):

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

The study quality assessment assisted us in making the data extraction form containing data needed to answer the research question. In this way, the quality evaluation was used in informing the different sections of data extraction. For example, once our quality criterion confirmed that there were comparison groups present in a particular study (according to criterion *D* in the Table 10 in Appendix A), the data extraction form used this information to extract *which* of the comparison groups were selected. Similarly the criterion *K* in the study quality assessment in the Table 10 in Appendix A allowed us to further extract data about the exact nature of the testing and training sets used.

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

3. Results

The 23 identified primary studies were related to the prediction and estimation of the following attributes:

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

Table 2 describes the relevant information regarding the included primary studies. The 23 primary studies were related to the application of GP for software quality classification (8 primary studies), software CES estimation (7 primary studies) and software fault prediction and software reliability growth modeling (8 primary studies).

Table 2
Distribution of primary studies per predicted/estimated attribute.

Domain	Ref.
SW quality classification (34.78%)	Robinson and McIlroy (1995) Khoshgoftaar et al. (2003) Liu and Khoshgoftaar (2001) Khoshgoftaar et al. (2004) Khoshgoftaar et al. (2004) Liu and Khoshgoftaar (2004) Reformat et al. (2003) Liu et al. (2006)
SW CES estimation (30.43%)	Dolado et al. (1998) Dolado (2000) Regolin et al. (2003) Dolado (2001) Burgess and Lefley (2001) Shan et al. (2002) Lefley and Shepperd (2003)
SW fault prediction and reliability growth (34.78%)	Kaminsky and Boetticher (2004) Kaminsky and Boetticher (2004) Tsakonas and Dounias (2008) Zhang and Chen (2006) Zhang and Yin (2008) Afzal and Torkar (2008) Costa et al. (2007) Costa and Pozo (2006)

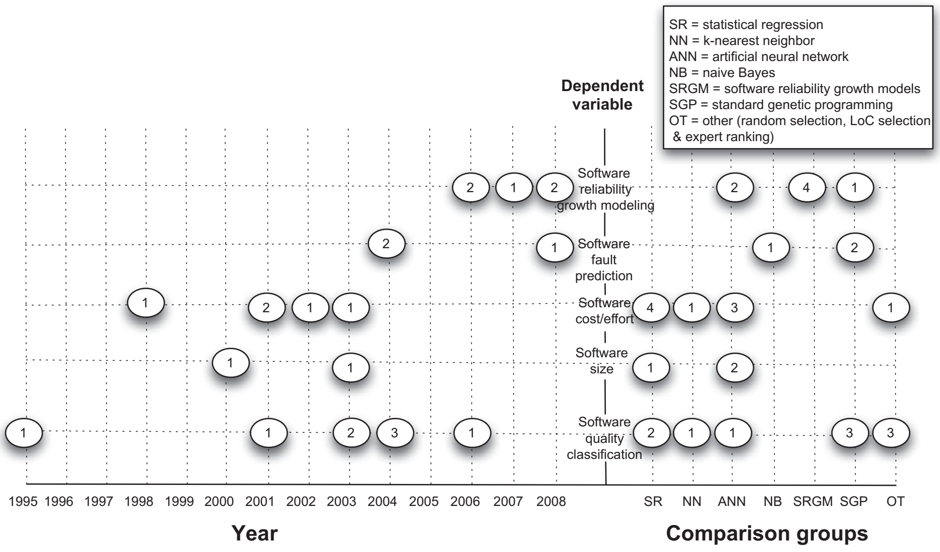

Fig. 1. Distribution of primary studies over range of applied comparison groups and time period.

Fig. 1 shows the year-wise distribution of primary studies within each category as well as the frequency of application of the different comparison groups. The bubble at the intersection of axes contains the number of primary studies. It is evident from the left division in this figure that the application of GP to prediction/estimation problems in software engineering has been scarce. This finding is perhaps little surprising; considering that the proponents of symbolic regression application of GP have highlighted several advantages of using it (Poli er al., 2007).

In the right division of Fig. 1, it is also clear that statistical regression techniques (linear, logistic, logarithmic, cubic, etc.) and artificial neural networks have been used as a comparison group for most of the studies.

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

3.1. Software quality classification

Our literature search found 8 studies on the application of symbolic regression using GP for software quality classification. Six out of these eight studies were co-authored by similar authors to a large extent, where one author was found to be part of each of these six studies. The data sets also over-lapped between studies which provides an indication that the conclusion of these studies were tied to the nature of the data sets used. However, these seven studies were marked with different variations of the GP fitness function and also used different comparison groups. This in our opinion indicates distinct contribution and thus worthy of inclusion as primary studies for this review. The importance of good fitness functions is also highlighted by Harman (2007): "*... no matter what search technique is employed, it is the fitness function that captures the crucial information; it differentiates a good solution from a poor one, thereby guiding the search.*"

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

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

Studies making use of GP for software quality classification argue that GP carries certain advantages for quality classification in comparison with traditional techniques because of its white-box and comprehensible classification model (Khoshgoftaar, Seliya, & Liu, 2003). This means that the GP models can potentially show the significant software metrics affecting the quality of modules. Additionally, by following a natural evolution process, GP can automatically extract the underlying relationships between the software metrics and the software quality, without relying on the assumption of the form and structure of the model.

In Robinson and McIlroy (1995), the authors use GP to identify fault-prone software modules. A software module is taken to comprise of a single source code file. Different software metrics were used as independent variables, with predictions assessed using five and nine independent variables. GP was compared with neural networks, *k*-nearest neighbor and linear regression. The methods were compared using two evaluation measures, accuracy and coverage. Accuracy was defined as the proportion of 'files predicted to be faulty' which were faulty, while coverage was defined as the proportion of 'files which were faulty' which were accurately predicted to be faulty. Using a measurement data corresponding to 163 software files, it was observed that in comparison with other techniques, GP results were reasonably accurate but lacked coverage.

Khoshgoftaar, Liu, and Seliya (2004) used a multi-objective fitness value that (i) maximized the best percentage of the actual faults averaged over the percentile level of interest (95%, 90%, 80%, 70% modules for quality enhancement) and (ii) restricted the size of the GP tree. The data set used in the study came from an embedded software system and five software metrics were used for quality prediction. The data set was divided into three random splits of the training and the testing data sets to avoid biased results. Based on the comparison of models ranked according to lines of code (LOC), the GP-models ranked the modules closer to the actual ranking on two of the three data splits. The results were not much different in an extension of this study (Khoshgoftaar, Liu, & Seliya, 2004), where in an additional case study of a legacy telecommunication system with 28 independent variables, GP outperformed the module ranking based on LOC.

Another study by Khoshgoftaar et al. (2003) used a different multi-objective fitness function for generating the software quality model. First the average weighted cost of misclassification was minimized and subsequently the trees were simplified by controlling their size. The average weighted cost of misclassification was formulated to penalize Type-II error (a *fp* module misclassified as *nfp*) more than Type-I error (a *nfp* module misclassified as *fp*). This was done by normalizing the cost of Type-II error with respect to the cost of Type-I error. Data was collected from two embedded systems applications which consisted of five different metrics for different modules. In comparison with standard GP, the performance of multi-objective GP was found to be better with multi-objective GP finding lower Type-I and Type-II error rates with smaller tree sizes. A similar study was carried out by Liu and Khoshgoftaar (2001) in which a single objective fitness function was used which took into account the average weighted cost of misclassification. Random subset selection was chosen which evaluated GP individuals in each generation on a randomly selected subset of the fit data set. Random subset selection helped to reduce the problem of over-fitting in GP solutions. Comparisons with logistic regression showed that Type-I and Type-II error rates for GP model were found to be better than for logistic regression. The same authors extended the study by adding a case study with data from a legacy telecommunication system in Liu and Khoshgoftaar (2004). This time the fitness function was multi-objective with minimization of expected cost of misclassification and also control of the tree size of GP solutions. The results of applying the random subset selection showed that over-fitting was reduced in comparison with when there was no random subset selection, hence, yielding solutions with better generalizability in the testing part of the data set.

In Reformat, Pedrycz, and Pizzi (2003), evolutionary decision trees were proposed for classifying software objects. The comparison group in this case was the classification done by two architects working on the project under study. The data set consisted of 312 objects whose quality was ranked by two architects as high, medium and low. The independent variables included 19 different software metrics for each object. Both genetic algorithms and GP were used to get best splitting of attribute domains for the decision tree and to get a best decision tree. The GA chromosome was represented by a possible splitting for all attributes. The fitness of the chromosome was evaluated using GP with two possibilities of the fitness function: (i) When the number of data samples in each class was comparable, $\frac{K}{N}$, where K= number of correctly specified data and N = number of data samples in a training set. (ii) When the number of data samples in each class were not comparable, $\prod_{i=1}^{c} \frac{k_i+1}{n_i}$, where c = number of different classes, n_i = number of data samples belonging to a class i and, finally, k_i = number of correctly classified data of a class i. The results showed that in comparison with architects' classification of objects' quality, the rate of successful classification for training data was around 66%–72% for the first and the second architect respectively.

In Liu, Khoshgoftaar, and Yao (2006), the performance of GP based software quality classification is improved by using a multi data set validation process. In this process, the hundred best models were selected after training on a single data set. These models were then validated on 5 *validation* data sets. The models that performed the best on these validation data sets were applied to the testing data set. The application of this technique to seven different NASA software projects showed that the misclassification costs were reduced in comparison with standard genetic programming solution.

Tables 3 and 4[1] show the relevant summary data extracted to answer the research question from each of the primary studies within software quality classification.

3.2. Software cost/effort/size (CES) estimation

In line with what Jørgensen and Shepperd suggest in Jørgensen and Shepperd (2007), we will use the term "cost" and "effort" interchangeably since effort is a typical cost driver in software development. We additionally take software size estimation to be related to either effort or cost and discuss these studies in this same section. According to Crespo et al. (2003), six different artificial intelligence methods are common in software development effort estimation. These are neural networks, case-based, regression trees, fuzzy logic, dynamical agents and genetic programming. We are here concerned with the application of symbolic regression using genetic programming as the base technique.

In Dolado et al. (1998), five different data sets were used to estimate software effort with line of code (LOC) and function points as the independent variables. Using the evaluation measures of pred (0.25) and MMRE (mean magnitude of relative error), it was observed that with respect to predictive accuracy, no technique was clearly superior. However, neural networks and GP were found to be flexible approaches as compared with classical statistics.

In Dolado (2000), different hypotheses were tested for estimating the size of the software in terms of LOC. Specifically, the component-based method was validated using three different techniques of multiple linear regression, neural networks and GP. Three different components were identified which included menus, input and reports. The independent variables were taken to be the number of choices within the menus and the number of data elements and relations for inputs and reports. For evaluating the component-based methodology in each project, six projects were selected having largest independent variables within each type of the component. Using the evaluation measures of MMRE and pred (0.25), it was observed that for linear relationships, small improvements obtained by GP in comparison with multiple linear regression came at the expense of the simplicity of the equations. However, it was also observed that the majority of the linear equations were rediscovered by GP. Also GP and neural networks (NN) showed superiority over multiple linear regression in case of non-linear relationship between the independent variables. The conclusion with respect to GP was that it provided similar or better values than regression equations and the GP solutions were also found to be transparent. Regolin, de Souza, Pozo, and Vergilio (2003) used a similar approach of estimating LOC from function points and number of components (NOC) metric. Using GP and NN, the prediction models using function points did not satisfy the criteria MMRE $\leqslant 0.25$ and pred $(0.25) \geqslant 0.75$. However, the prediction models for estimating lines of code from NOC metric were acceptable from both the NN and the GP point of view.

In Dolado (2001), genetic programming and different types of standard regression analysis (linear, logarithmic, inverse quadratic, cubic, power, exponential, growth and logistic) were used to find a relationship between software size (independent variable) and cost (dependent variable). The predictive accuracy measures of pred (0.25) and MMRE showed that linear regression consistently obtained the best predictive values, with GP achieving a significant improvement over classical regression in 2 out of 12 data sets. GP performed well, pred (0.25), on most of the data sets but

[1] The data sets in Table 4 are taken at a coarser level, e.g. ISBSG data (ISBSG, 2009) of multiple projects is 1 data set.

Table 3
Summary data for primary studies on GP application for software quality classification. (?) indicates absence of information and ($\sim$) indicates indifferent results.

Article	Dependent variable	Fitness function	Comparison group	Evaluation measures	GP better?
Robinson and McIlroy (1995)	Fault proneness based on number of faults	Minimization of root mean square	Neural networks, *k*-nearest neighbor, linear regression	Accuracy & coverage	$\sim$
Khoshgoftaar et al. (2003)	Fault proneness based on number of faults	Minimization of average cost of misclassification and minimization of tree size	Standard GP	Type I, Type II and overall error rates	√
Liu and Khoshgoftaar (2001)	Fault proneness based on number of faults	Minimization of the average cost of misclassification	Logistic regression	Type I, Type II and overall error rates	√
Khoshgoftaar et al. (2004)	Number of faults for each software module	Maximization of the best percentage of actual faults averaged over the percentiles level of interest and controlling the tree size	Ranking based on lines of code	Number of faults accounted by different cut-off percentiles	√
Khoshgoftaar et al. (2004)	Number of faults for each software module	Maximization of the best percentage of actual faults averaged over the percentiles level of interest and controlling the tree size	Ranking based on lines of code	Number of faults accounted by different cut-off percentiles	√
Liu and Khoshgoftaar (2004)	Fault proneness based on number of faults	Minimization of expected cost of misclassification and controlling the tree size	Standard GP	Number of over-fitting models and Type I, Type II error rates	√
Reformat et al. (2003)	Ranking of object's quality	(a) $\frac{K}{N}$ (b) $\prod_{i=1}^{c}\frac{k_i+1}{n_i}$	Quality ranking of an object assessed by the architects	Rate of successful classification for training and testing set	$\sim$
Liu et al. (2006)	Fault proneness based on number of faults	Minimization of the expected cost of misclassification and controlling the tree size	Standard GP	Type I and Type II error rates	√

Table 4
Data set characteristics for primary studies on GP application for software quality classification. (?) indicates absence of information.

Article	Data sets No.	Sampling of training and testing sets	Industrial (I) or academic (A)	Data sets public or private
Robinson and McIlroy (1995)	1	103 records for training and 60 records for testing	?	Private
Khoshgoftaar et al. (2003)	1	Approximately $\frac{2}{3}$ for training and the rest for testing	I	Private
Liu and Khoshgoftaar (2001)	1	Approximately $\frac{2}{3}$ for training and the rest for testing and random subset selection	I	Private
Khoshgoftaar et al. (2004)	1	$\frac{2}{3}$ for training and the rest for testing, three splits	I	Private
Khoshgoftaar et al. (2004)	2	$\frac{2}{3}$ for training and the rest for testing, three splits	I	Private
Liu and Khoshgoftaar (2004)	1	Training on release 1 data set, testing on release 2,3,4 data sets	I	Private
Reformat et al. (2003)	1	10-fold cross-validation	I	Private
Liu et al. (2006)	7	1 training data set, 5 validation data sets and 1 testing data set	I	Private

sometimes at the expense of MMRE. This also indicated the potential existence of over-fitting in GP solutions. It was also found that size alone as an independent variable for predicting software cost is not enough since it did not define the types of economies of scale or marginal return with clarity.

The study by Burgess and Lefley (2001) extends the previous study from Dolado (2001) by using 9 independent variables to predict the dependent variable of effort measured in person hours. Using the Desharnais data set of 81 software projects, the study showed that GP is consistently more accurate for MMRE but not for adjusted mean square error (AMSE), pred (0.25) and balanced mean magnitude of relative error (BMMRE). The study concluded that while GP and NN can provide better accuracy, they required more effort in setting up and training.

In Shan, McKay, Lokan, and Essam (2002) the authors used grammar-guided GP on 423 projects from release 7 of the ISBSG (The International Software Benchmarking Standards Group Limited (ISBSG, 2009)) data set to predict software project effort. The evaluation measures used were *R*-squared, MSE, MMRE, pred (0.25) and pred (0.5). In comparison with linear and log–log regression, the study showed that GP was far more accurate than simple linear regression. With respect to MMRE, log–log regression was better than GP which led to the conclusion that GP maximizes one evaluation criterion at the expense of the other. The study showed that grammar guided GP provides both a way to represent syntactical constraints on the solutions and a mechanism to incorporate domain knowledge to guide the search process.

Lefley and Shepperd (2003) used several independent variables from 407 cases to predict the total project effort comparing GP, ANN, least squares regression, nearest neighbor and random selection of project effort. With respect to the accuracy of the predictions, GP achieved the best level of accuracy the most often, although GP was found hard to configure and the resulting models could be more complex.

Tables 5 and 6[2] present the relevant summary data extracted to answer the research question from each of the primary studies within software CES estimation.

[2] The data sets in Table 6 are taken at a coarser level, e.g. ISBSG data (ISBSG, 2009) of multiple projects is 1 data set.

Table 5
Summary data for primary studies on GP application for software CES estimation. (~) indicates indifferent results.

Article	Dependent variable	Fitness function	Comparison group	Evaluation measures	GP better?
Dolado et al. (1998)	Software effort	Mean squared error	Neural networks & linear regression	pred (0.25)[a] and MMRE[b]	~
Dolado (2000)	Software size	Mean squared error	Neural networks & multiple linear regression	pred (0.25) and MMRE	~
Regolin et al. (2003)	Software size	MMRE	Neural networks	pred (0.25) and MMRE	~
Dolado (2001)	Software cost	Mean square error	Linear, logarithmic, inverse quadratic, cubic, power, exponential, growth and logistic regression	pred (0.25) and MMRE	~
Burgess and Lefley (2001)	Software effort	MMRE	Neural networks	MMRE, AMSE[c], pred (0.25), BMMRE[d]	~
Shan et al. (2002)	Software effort	Mean square error	Linear regression, log–log regression	*R*-squared[e], MMRE, pred (0.25) and pred (0.5)	~
Lefley and Shepperd (2003)	Software effort	?	ANN, least squares regression, nearest neighbor and random selection of project effort	Pearson correlation coefficient of actual and predicted, AMSE, pred (0.25), MMRE, BMMRE, worst case error, the ease of set up and the explanatory value	~

[a] Prediction at level 0.25.
[b] Mean magnitude of relative error.
[c] Adjusted mean magnitude of relative error.
[d] Balanced mean magnitude of relative error.
[e] Coefficient of multiple determination.

3.3. Software fault prediction and reliability growth

Apart from studies on software quality classification (Section 3.1), where the program modules are classified as being either *fp* or *nfp*, there are studies which are concerned with prediction of either the fault content or software reliability growth.

In Kaminsky and Boetticher (2004) the authors proposed the incorporation of existing equations as a way to include domain knowledge for improving the standard GP algorithm for software fault prediction. They specifically used Akiyama's equations (Akiyama, 1971), Halstead's equation (Halstead, 1977), Lipow's equation (Lipow, 1982), Gaffney's equation (Gaffney, 1984) and Compton's equation (Compton & Withrow, 1990) to add domain knowledge to a simple GP algorithm which is based on mathematical operators. Using the fitness function (1-standard error), six experiments were performed using a NASA data set of 379 C functions. Five of these experiments compared standard GP with GP enhanced with Akiyama's, Halstead's, Lipow's, Gaffney's and Compton's equations. The last experiment compared standard GP with GP enhanced with all these equations simultaneously. The results showed that by including explicit knowledge in the GP solutions, the fitness values for the GP solutions increased.

In another study, Kaminsky and Boetticher (2004), the same authors used another approach called data equalization to compensate for data skewness. Specifically, duplicates of interesting training instances (in this case functions with greater than zero faults) were added to the training set until the total reached the frequency of most occurring instance (in this case functions with zero faults). The fitness function used was: $1 + e^{(7*(1-n-k)/(n-1)*Se^2/Sy^2)}$, where k = number of inputs, n = number of valid results, Se = standard error and Sy = standard deviation. Using the same data sets as before, the experimental results showed that the average fitness values for the equalized data set were better than for the original data set.

In Tsakonas and Dounias (2008), grammar-guided GP was used on NASA's data set consisting of four projects to measure the probability of detection, PD (the ratio of faulty modules found to all known faulty modules) and false alarm rate, PF (the ratio of number of non-faulty modules misclassified as faulty to all known non-faulty modules). The fitness function represented the coverage of knowledge represented in the individual, and equaled $\frac{tp}{(tp+fn)} * \frac{tn}{(tn+fp)}$ where fp is the number of false positives, tp the number of true positives, tn the number of true negatives and fn the number of false negatives. The study showed that grammar-guided GP performed better than naive Bayes on both measures (PD and PF) in two of the projects' data while in the rest of the two data, it was better in one of the two measures.

We were also able to find a series of studies where the comparison group included traditional software reliability growth models. Zhang and Chen (2006) used mean time between failures (MTBF) time series to model software reliability growth using genetic programming, neural networks (NN) and traditional software reliability models, i.e. Schick–Wolverton, Goel–Okumoto, Jelinki–Moranda and Moranda. Using multiple evaluation measures of short-term range error, prequential likelihood, model bias, model bias trend, goodness of fit and model noise; the GP approach was found better than the traditional software reliability growth models. However, it is not clear from the study how neural networks performed against all the evaluation measures (except for the short-term range error where GP was better than neural networks). Also it is not clear from the study what sampling strategy was used to split the data set into training and testing set. The fitness function information is also lacking from the study. The study is however extended in Zhang and Yin (2008) with adaptive crossover and mutation probabilities, and the corresponding GP was named adaptive genetic programming. In comparison with standard GP and the same reliability growth models (as used in the previous study), the mean time between failures (MTBF) and the next mean time to failure (MTTF) values for adaptive GP were found to be more accurate.

Afzal and Torkar (2008) used fault data from three industrial software projects to predict the software reliability in terms of number of faults. Three traditional software reliability growth models (Goel–Okumoto, Brooks and Motley, and Yamada's *S*-shaped) were chosen for comparison using the fitness function of sum of absolute differences between the obtained and expected results in all fitness cases, $\sum_{i=1}^{n}|e_i - e_i'|$, where e_i is the actual fault count data, e_i' the estimated value of the fault count data and n

Table 6
Data set characteristics for primary studies on GP application for software CES estimation.

Article	Data sets No.	Sampling of training and testing sets	Industrial (I) or academic (A)	Data sets public or private
Dolado et al. (1998)	5	(a) Train and test a model with all the points. (b) Train a model on 66% of the data points and test on 34% of the points	I	Public & Private
Dolado (2000)	6	Train a model on 60–67% of the data points and test in 40–37%	A	Public
Regolin et al. (2003)	2	Train on $\frac{2}{3}$ and test on $\frac{1}{3}$	I&A	Public
Dolado (2001)	12	Training and testing on all data points	I&A	Public
Burgess and Lefley (2001)	1	Training on 63 projects, testing on 18 projects	I	Public
Shan et al. (2002)	1	Random division of 50% in training set and 50% in testing set	I	Public
Lefley and Shepperd (2003)	1	149 projects in the training set and 15 projects in the testing set	I	Public

Table 7
Summary data for primary studies on GP application for software fault prediction and reliability growth. (?) indicates absence of information and (~) indicates indifferent results.

Article	Dependent variable	Fitness function	Comparison group	Evaluation measures	GP better?
Kaminsky and Boetticher (2004)	Software fault prediction	1-standard error	Standard GP	Fitness values	√
Kaminsky and Boetticher (2004)	Software fault prediction	$1 + e^{(7*(1-n-k)/(n-1)*Se^2/Sy^2)}$	Standard GP	Fitness values	√
Tsakonas and Dounias (2008)	Software fault prediction	$\frac{tp}{(tp+fn)} * \frac{tn}{(tn+fp)}$	Naive Bayes	PD & PF	~
Zhang and Chen (2006)	Software reliability	?	Neural networks and traditional software reliability growth models	Short-term range error, prequential likelihood, model bias, model bias trend, goodness of fit and model noise	√
Zhang and Yin (2008)	Software reliability	?	Traditional software reliability growth models	Mean time between failures and next mean time to failure	√
Afzal and Torkar (2008)	Software reliability	$\sum_{i=1}^{n} \lvert e_i - e_i' \rvert$	Traditional software reliability growth models	Prequential likelihood ratio, AMSE, Braun statistic, Kolmogorov–Smirnov test and distribution of residuals	√
Costa et al. (2007)	Software reliability	WRMSE[a]	Traditional software reliability growth models and ANN	Maximum deviation, average bias, average error, prediction error, correlation coefficient, Kolmogorov–Smirnov	√
Costa and Pozo (2006)	Software reliability	RMSE[b]	Standard GP	Maximum deviation, average bias, average error, prediction error and correlation coefficient	√

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

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

In Costa et al. (2007), the authors used GP and GP with boosting to model software reliability. The comparisons were done with time based reliability growth models (Jelinski–Moranda and geometric), coverage-based reliability growth model (coverage-based binomial model) and artificial neural network (ANN). The evaluation measures used for time-based models included maximum deviation, average bias, average error, prediction error and correlation coefficient. For coverage-based models, an additional Kolmogorov–Smirnov test was also used. The fitness function used was weighted root mean square error (WRMSE), $\sqrt{\sum_{i=1}^{m}(x_i - x_i^d)^2 D_i m}$ where x_i = real value, x_i^d = estimated value, D_i = weight of each example and m = size of the data set. Using the first $\frac{2}{3}$ of the data set as a training set, it was observed that GP with boosting (GPB) performed better than traditional models for models based on time. However, there was no statistical difference between GP, GPB and ANN models. For models based on test coverage, the GPB models' results were found to be significantly better compared to that of the GP and ANN models.

In Costa and Pozo (2006), the authors used a modified GP algorithm called the $\mu + \lambda$ GP algorithm to model software reliability growth. In the modified algorithm, n% of the best individuals were applied the genetic operators in each generation. The genetic operators generated λ individuals, which competed with their parents in the selection of μ best individuals to the next generation where ($\lambda > \mu$). The fitness function used was root mean square error (RMSE), given by: $\sqrt{\frac{\sum_{i=1}^{n}\lvert x_i - x_i^d \rvert}{n}}$ where x_i is the real value, x_i^d is the estimated value and n is the size of the data set. Using measures as maximum deviation, average bias, average error, prediction error and correlation coefficient; the results favored modified GP algorithm. Additional paired two-sided t-tests for average error

Table 8
Data set characteristics for primary studies on GP application for software fault prediction and reliability growth. (?) indicates absence of information.

Article	Data sets No.	Sampling of training and testing sets	Industrial (I) or academic (A)	Data sets public or private
Kaminsky and Boetticher (2004)	1	?	I	Public
Kaminsky and Boetticher (2004)	1	?	I	Public
Tsakonas and Dounias (2008)	1	10-fold cross-validation	I	Public
Zhang and Chen (2006)	1	?	I	Private
Zhang and Yin (2008)	1	?	I	Private
Afzal and Torkar (2008)	3	First $\frac{2}{3}$ of the data set for training and the rest for testing	I	Private
Costa et al. (2007)	2	First $\frac{2}{3}$ of the data set for training and the rest for testing	I	Public & Private
Costa and Pozo (2006)	1	First $\frac{2}{3}$ of the data set for training and the rest for testing	I	Public

confirmed the results in favor of modified GP with a statistically significant difference in the majority of the results between the modified and standard GP algorithm.

Tables 7 and 8[3] shows the relevant summary data extracted to answer the research question from each of the primary studies within software fault prediction and reliability growth.

4. Discussion and areas of future research

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

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

For software quality classification, six out of eight studies reported results in favor of using GP for the classification task. Two studies were inconclusive in favoring a particular technique either because the different measures did not converge, as in Robinson and McIlroy (1995), or the proposed technique used GP for initial investigative purposes only, without being definitive in the judgement of GP's efficacy, as in Reformat et al. (2003) (these two studies are indicated by the sign $\sim$ in Table 3).

The other six studies were co-authored by similar authors to a large extent and the data sets also over-lapped between studies but these studies contributed in introducing different variations of the GP fitness function and also used different comparison groups. These six studies were in agreement that GP is an effective method for software quality classification based on comparisons with neural networks, *k*-nearest neighbor, linear regression and logistic regression. Also GP was used to successfully rank-order software modules in a better way than the ranking done on the basis of lines of code. Also it was shown that numerous enhancements to the GP algorithm are possible hence improving the evolutionary search in comparison with standard GP algorithm. These enhancements include random subset selection and different mechanisms to control excessive code growth during GP evolution. Improvements to the GP algorithm gave better results in comparison with standard GP algorithm for two studies (Khoshgoftaar et al., 2003; Liu & Khoshgoftaar, 2004). However, one finds that there can be two areas of improvement in these studies: (i) Increasing the comparisons with more techniques. (ii) Increasing the use of public data sets.

We also observe from Table 3 that multi-objective GP is an effective way to seek near-optimal solutions for software quality classification in the presence of competing constraints. This indicates that further problem-dependent objectives can possibly be represented in the definition of the fitness function which potentially can give better results. We also believe that in order to generalize the use of GP for software quality classification, the comparison groups need to increase.

There are many different techniques that have been applied by researchers to software quality classification, see e.g. (Lessmann, Baesens, Mues, & Pietsch, 2008). GP needs to be compared with a more representative set of techniques that have been found successful in earlier research—only then are we be able to ascertain that GP is a competitive technique for software quality classification. We see from Table 4 that all the data sets were private. In this regards, the publication of private data sets needs to be encouraged. Publication of data sets would encourage other researchers to replicate the studies based on similar data sets and hence we can have greater confidence in the correctness of the results. Another aspect that requires improvement is to include statistical hypothesis testing that is currently very much ignored in studies of software quality classification. Nevertheless, one encouraging trend that is observable from Table 4 is that the data sets represented real world projects which adds to the external validity of these results.

For software CES estimation, there was no strong evidence of GP performing consistently on all the evaluation measures used (as shown in Table 5). The sign $\sim$ in the last column of Table 5 shows that the results are inconclusive concerning GP. The study results indicate that while GP scores higher on one evaluation measure, it lags behind on others. There is also a trade-off between different qualitative factors, e.g. complexity of interpreting the end solution, and the ease of configuration and flexibility to cater for varying data sets. The impression from these studies is that GP also requires some effort in configuration and training. There can be different reasons related to the experimental design for the inconsistent results across the studies using GP for software CES estimation. One reason is that the accuracy measures used for evaluation purposes are not near to a standardized use. While the use of pred (0.25) and MMRE are commonly used, other measures, including AMSE and BMMRE, are also applied. It is important that researchers are aware of the merits/demerits of using these evaluation measures (Foss, Stensrud, Kitchenham, & Myrtveit, 2003; Shepperd, Cartwright, & Kadoda, 2000). Another aspect which differed between the studies was the sampling strategies used for training and testing sets (Column 3, Table 6). These different sampling strategies are also a potential contributing factor in inconsistent model results. As with the studies on software quality classification, statistical hypothesis testing needs to be an essential part of the study design for software CES estimation. What is also observable from these studies is that over-fitting is a common

[3] The data sets in Table 8 are taken at a coarser level, e.g. ISBSG data (ISBSG, 2009) of multiple projects is 1 data set.

Table 9
Summary of the studies showing inconclusive results in using GP.

Article	Quotation
Robinson and McIlroy (1995)	While generally not as good as the results obtained from other methods, the GP results are reasonably accurate but low on coverage
Reformat et al. (2003)	The rate of successful classifications for training data is around 66 and 72% for the first architect and the second architect, respectively. In the case of testing data the rates are 55 and 63%
Dolado et al. (1998)	However, from the point of view of making good predictions, no technique has been proved to be clearly superior . . . From the values shown in the tables, there is no great superiority of one method versus the others . . . GP can be used as an alternative to linear regression, or as a complement to it.
Dolado (2000)	The final impression is that GP has worked very well with the data used in this study. The equations have provided similar or better values than the regression equations. Furthermore, the equations are "intelligible", providing confidence in the results. . . . In the case of linear relationships, some of the small improvements obtained by GP compared to MLR come at the expense of the simplicity of the equations, but the majority of the linear equations are rediscovered by GP
Regolin et al. (2003)	We cannot conclude GP is a better technique than NN . . . GP and ANN are valid and promising approaches to size estimation . . . However, GP presents additional advantages with respect to NN. The main advantage of using GP is the easy interpretation of result
Dolado (2001)	From the point of view of the capabilities of the two methods, GP achieves better values in the pred (0.25) in eleven out of the twelve data sets, but sometimes at the cost of having a slight worse value of the MMRE. Only in data sets A and H, GP provides a significant improvement over classical regression
Burgess and Lefley (2001)	There is evidence that GP can offer significant improvements in accuracy but this depends on the measure and interpretation of accuracy used. GP has the potential to be a valid additional tool for software effort estimation but set up and running effort is high and interpretation difficult . . .
Shan et al. (2002)	Log regression models perform much worse than GP on MSE, about the same as GP on R^2 and pred (0.25), and better than GP on MMRE and pred (0.5). One way of viewing this is that GP has more effectively fit the objective, namely minimizing MSE, at the cost of increased error on other measures
Lefley and Shepperd (2003)	The results do not find a clear winner but, for this data, GP performs consistently well, but is harder to configure and produces more complex models
Tsakonas and Dounias (2008)	In two of the databases, our model is proved superior to the existing literature in both comparison variables, and in the rest two databases, the system is shown better in one of the two variables

problem for GP. However, there are different mechanisms to avoid over-fitting, such as random subset selection on the training set and placing limits on the size of the GP trees. These mechanisms should be explored further.

As previously pointed out in Section 3.2, Crespo et al. (2003) identified six artificial intelligence techniques applicable to software development effort estimation. It is interesting to note that our literature search did not find any study that compares all of these techniques.

As for the studies related to software fault prediction and software reliability growth, seven out of eight studies favor the use of GP in comparison with neural networks, naive Bayes and traditional software reliability growth models (this is evident from the last column in Table 7). However, as Table 8 showed, it was not clear from four studies which sampling strategies were used for the training and testing sets. From two of these four studies, it was also not clear what fitness function was used for the GP algorithm. If, however, we exclude these four studies from our analysis, GP is still a favorable approach for three out of four studies. With respect to comparisons with traditional software reliability growth models, the main advantage of GP is that it is not dependent on the assumptions that are common in these software reliability growth models. Also GP promises to be a valid tool in situations where different traditional models have inconsistent results. Besides, we also observe that several improvements to the standard GP algorithm are possible which provides comparatively better results. Specifically, we see studies where the incorporation of explicit domain knowledge in the GP modeling process has resulted in improved fitness values (Kaminsky & Boetticher, 2004). Table 7 also shows that the variety of comparison groups is represented poorly; there is an opportunity to increase the comparisons with more techniques and also to use a commonly used technique as a baseline.

For studies which were inconclusive in the use of GP for prediction/estimation, we include quotations from the respective papers in Table 9 (an approach similar to the one used in Mair & Shepperd (2005)) that reflects the indifference between GP and other approaches. What is evident from these studies is the following:

1. The accuracy of GP as a modeling approach is attached to the evaluation measure used. The impression from these studies is that GP performs superior on one evaluation measure at the cost of the other. This indicates that the GP fitness function should not only be dependent on the minimization of standard error but also biased in searching those solutions which reflect properties of other evaluation measures, such as correlation coefficient.
2. The qualitative scores for GP models are both good and bad. While they might be harder to configure and result in complex solutions, the results can nevertheless be interpreted to some extent. This interpretation can be in the form of identifying the few significant variables (Kotanchek et al., 2003). But another key question is that whether or not we are able to have a reasonable explanation of the relationship between the variables. As an example, Dolado (2000) provides the following equation generated by GP:

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

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

Based on the above discussion, we can conclude that while the use of GP as a prediction tool has advantages, as presented in Section 3, there are, at the same time, challenges to overcome as points 1 and 2 indicate above. We believe that these challenges offer promising future work to undertake for researchers.

5. Validity threats

We assume that our review is based on studies which were unbiased. If this is not the case, then the validity of this study is also expected to suffer (Mair & Shepperd, 2005). Also, like any other systematic review, this one too is limited to making use of information given in the primary studies (Kitchenham et al., 2007). There is also a threat that we might have missed a relevant study but we are confident that both automated and manual searches of the key information sources (Section 2.2) have given us a complete set. Our study selection procedure (Section 2.3) is straightforward and the researchers had agreement on which studies to include/exclude. However, this review does not cover unpublished research that had undesired outcome and company confidential results.

6. Conclusions

This systematic review investigated whether symbolic regression using genetic programming is an effective approach in comparison with machine learning, regression techniques and other competing methods (including different improvements over the standard GP algorithm). The results of this review resulted in a total of 23 primary studies; which were further classified into software quality classification (eight studies), software CES estimation (seven studies) and fault prediction/software reliability growth (eight studies).

Within software quality classification, we found that in six out of eight studies, GP performed better than competing techniques (i.e. neural networks, *k*-nearest neighbor, linear regression and logistic regression). Different enhancements to the standard GP algorithm also resulted in more accurate quality classification, while GP was also more successful in rank-ordering of software modules

Table 10
Study quality assessment.

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

	Robinson and McIlroy (1995)	Khoshgoftaar et al. (2003)	Liu and Khoshgoftaar (2001)	Khoshgoftaar et al. (2004)	Khoshgoftaar et al. (2004)	Liu and Khoshgoftaar (2004)	Reformat et al. (2003)	Liu et al. (2006)
(b) Study quality assessment for software quality classification studies								
A	√	√	√	√	√	√	√	√
B	√	√	√	√	√	√	√	√
C	~×	~√	~√	~√	~√	~√	~√	~√
D	√	√	√	√	√	√	√	√
E	~√	~√	~√	~√	~√	~√	~√	
F	√	√	√	√	√	√	√	√
G	×	×	×	×	×	×	×	×
H	√	√	√	√	√	√	√	√
I	√	√	√	√	√	√	√	√
J	~√	~√	√	√	√	~√	×	√
K	√	√	√	√	√	√	√	√

	Dolado et al. (1998)	Dolado (2000)	Regolin et al. (2003)	Dolado (2001)	Burgess and Lefley (2001)	Shan et al. (2002)	Lefley and Shepperd (2003)
(c) Study quality assessment for software CES estimation							
A	√	√	√	√	√	√	√
B	√	√	√	√	√	√	√
C	√	~√	√	√	√	√	√
D	√	√	√	√	√	√	√
E	~√	√	√	√	√	√	√
F	√	√	√	√	√	√	√
G	×	×	×	×	~√	×	×
H	√	√	√	√	√	√	√
I	√	√	√	√	√	√	√
J	√	√	√	√	√	√	√
K	√	√	√	√	√	√	√

	Kaminsky and Boetticher (2004)	Kaminsky and Boetticher (2004)	Tsakonas and Dounias (2008)	Zhang and Chen (2006)	Zhang and Yin (2008)	Afzal and Torkar (2008)	Costa et al. (2007)	Costa and Pozo (2006)
(d) Study quality assessment for software fault prediction and software reliability growth modeling								
A	√	√	√	√	√	√	√	√
B	√	√	√	√	√	√	√	√
C	~√	~√	~√	~√	~√	√	√	~ √
D	√	√	√	√	√	√	√	√
E	√	√	√	×	×	~√	~√	√
F	√	√	√	~√	~√	√	√	√
G	√	√	×	×	×	√	√	√
H	√	√	√	√	√	√	√	√
I	√	√	√	~√	~√	√	√	√
J	~√	~√	√	~ √	~√	√	√	√
K	×	×	√	×	×	√	√	√

in comparison with random ranking and ranking based on lines of code. We concluded that GP seems to be an effective method for software quality classification. This is irrespective of the fact that one author was part of seven out of nine primary studies and the fact that there was an overlap of data sets used across the studies. This is because we considered each of these primary studies representing a distinct contribution in terms of different algorithmic variations.

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

The results for software fault prediction and software reliability growth modeling leaned towards the use of GP, with seven out of eight studies resulting in GP performing better than neural networks, naive Bayes and traditional software reliability growth models. Although four out of these eight studies lacked in some of the quality instruments used in Table 10 (Appendix A); still three out of the remaining four studies reported results in support of GP. We therefore concluded that the current literature provides evidence in support of GP being an effective technique for software fault prediction and software reliability growth modeling.

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

1. Use public data sets wherever possible. In case of private data sets, there are ways to transform the data sets to make it public domain (e.g., one such transformation is discussed in Wood (1996)).
2. Apply commonly used sampling strategies to help other researchers replicate, improve or refute the established predictions and estimations. From our sample of primary studies, the sampling strategy of $\frac{2}{3}$ for training, remaining $\frac{1}{3}$ for testing and 10-fold cross validation are mostly used. Kitchenham et al. (2007) recommends using a jackknife approach with leave-one-out cross-validation process for smaller data sets; this needs to be validated further.
3. Avoiding over-fitting in GP solutions is possible and is beneficial to increase the generalizability of model results in the testing data set. The primary studies in this review offer important results in this regard.
4. Always report the settings used for the algorithmic parameters (also suggested in Barr, Golden, Kelly, Resende, & Junior (1995)).
5. Compare the performances against a comparison group that is both commonly used and currently an active field of research. For our set of primary studies, comparisons against different forms of statistical regression and artificial neural networks can be seen as a baseline for comparisons.
6. Use statistical experimental design techniques to minimize the threat of differences being caused by chance alone (Myrtveit & Stensrud, 1999).
7. Report the results even if there is no statistical difference between the quality of the solutions produced by different methods (Barr et al., 1995).

Acknowledgements

The authors thank Claes Wohlin for his comments on the earlier drafts of the paper.

Appendix A. Study quality assessment

See Table 10.

References

Abraham, A. (2008). Real time intrusion prediction, detection and prevention programs. In *2008 IEEE international conference on intelligence and security informatics (ISI'08), Piscataway, NJ, USA*.
Afzal, W., & Torkar, R. (2008). A comparative evaluation of using genetic programming for predicting fault count data. In *Proceedings of the 3rd international conference on software engineering advances (ICSEA'08), Piscataway, NJ, United States*.
Afzal, W., Torkar, R., & Feldt, R. (2009). A systematic review of search-based testing for non-functional system properties. *Information and Software Technology, 51*(6), 957–976.
Akiyama, F. (1971). An example of software system debugging. *International Federation for Information Processing Congress, 71*(1), 353–359.
Alander, J. T. (1995). An indexed bibliography of genetic programming, Report Series no 94-1-GP, Department of Information Technology and Industrial Management, University of Vaasa, Finland, last checked: 13 Feb 2009 (1995). <ftp://ftp.uwasa.fi/cs/report94-1/gaGPbib.ps.Z>.
Alfaro-Cid, E., McGookin, E. W., Murray-Smith, D. J., & Fossen, T. I. (2008). Genetic programming for the automatic design of controllers for a surface ship. *IEEE Transactions on Intelligent Transportation Systems, 9*(2), 311–321.
Andersson, B., Svensson, P., Nordin, P., & Nordahl, M. (1999). Reactive and memory-based genetic programming for robot control. In *Proceedings of the 2nd European workshop on genetic programming (EuroGP'99), Berlin, Germany*.
Bäck, T., Fogel, D. B., & Michalewicz, Z. (Eds.). (2000). *Evolutionary computation 1 – Basic algorithms and operators*. New York, USA: Taylor & Francis Group, LLC.
Banzhaf, W., Nordin, P., Keller, R., & Francone, F. (1998). *Genetic programming – An introduction*. Morgan Kaufmann Publishers, Inc..
Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., & Junior, W. R. S. (1995). Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics, 1*(1), 9–32.
Burgess, C. J., & Lefley, M. (2001). Can genetic programming improve software effort estimation? a comparative evaluation. *Information and Software Technology, 43*(14), 863–873.
Burke, E. K., & Kendall, G. (2005). In E K. Burke & G. Kendall (Eds.), *Search methodologies – Introductory tutorials in optimization and decision support techniques*. New York, USA: Springer Science and Business Media, Inc. [Chapter 1 – Introduction].
Catal, C., & Diri, B. (2009). A systematic review of software fault prediction studies. *Expert Systems with Applications, 36*(4), 7346–7354.
Compton, B. T., & Withrow, C. (1990). Prediction and control of Ada software defects. *Journal of Systems and Software, 12*(3), 199–207.
Costa, E. O., de Souza, G. A., Pozo, A. T. R., & Vergilio, S. R. (2007). Exploring genetic programming and boosting techniques to model software reliability. *IEEE Transactions on Reliability, 56*(3), 422–434.
Costa, E. O., & Pozo, A. (2006). A mu + lambda – GP algorithm and its use for regression problems. In *Proceedings of the 18th IEEE international conference on tools with artificial intelligence (ICTAI'06)*. Washington, DC, USA: IEEE Computer Society.
Costa, E. O., Pozo, A., & Vergilio, S. R. (2006). Using boosting techniques to improve software reliability models based on genetic programming. In *Proceedings of the 18th IEEE international conference on tools with artificial intelligence (ICTAI'06)*. Washington, D.C, USA: IEEE Computer Society.
Costa, E. O., Vergilio, S. R., Pozo, A. T. R., & de Souza, G. A. (2005). Modeling software reliability growth with genetic programming. In *Proceedings of the 16th international symposium on software reliability engineering (ISSRE'05)*. Chicago, IL, USA: IEEE Computer Society.
Crespo, J., Cuadrado, J. J., Garcia, L., Marban, O., &Sanchez-Segura, M. I. (2003). Survey of artificial intelligence methods on software development effort estimation. In *Proceedings of the 10th ISPE international conference on concurrent engineering, Swets en Zeitlinger B.V*.
de Almeida, M. A., Lounis, H., & Melo, W. L. (1998). An investigation on the use of machine learned models for estimating correction costs. In *Proceedings of the 20th international conference on software engineering (ICSE'98)*.
Dohi, T., Nishio, Y., & Osaki, S. (1999). Optimal software release scheduling based on artificial neural networks. *Annals of Software Engineering, 8*, 167–185.
Dolado, J. J. (2000). A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering, 26*(10), 1006–1021.
Dolado, J. J. (2001). On the problem of the software cost function. *Information and Software Technology, 43*(1), 61–72.
Dolado, J. J., & Fernandez, L. (1998). Genetic programming, neural networks and linear regression in software project estimation. In *Proceedings of the international conference on software process improvement, research, education and training (INSPIRE'98)*. London: British Computer Society.
Dybå, T., Dingsøyr, T., & Hanssen, G. K. (2007). Applying systematic reviews to diverse study types: An experience report. In *Proceedings of the 1st international symposium on empirical software engineering and measurement (ESEM'07)*.
Evett, M., Khoshgoftar, T., der Chien, P., & Allen, E. (1998). GP-based software quality prediction. In *Proceedings of the 3rd annual genetic programming conference*.

Foss, T., Stensrud, E., Kitchenham, B., & Myrtveit, I. (2003). A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering, 29*(11), 985–995.

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

Halstead, M. H. (1977). *Elements of software science*. North-Holland: Elsevier.

Harman, M. (2007). The current state and future of search-based software engineering. In *Proceedings of future of software engineering at 29th international conference on software engineering (FOSE'07)*. USA: IEEE Computer Society.

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

ISBSG, The International Software Benchmarking Standards Group Limited, last checked: 18 Mar 2009 (2009). <http://www.isbsg.org/>.

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

Jørgensen, M. (2009). BESTweb, a repository of software cost and effort estimation papers – Simula research laboratory, last checked: 07 Mar 2009. <http://home.simula.no/BESTweb/>.

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

Kaminsky, K., & Boetticher, G. (2004). Building a genetically engineerable evolvable program (GEEP) using breadth-based explicit knowledge for predicting software defects. In *Proceedings of the 2004 IEEE annual meeting of the fuzzy information processing (NAFIPS'04)*.

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

Khoshgoftaar, T. M., Seliya, N., Liu, Y. (2003). Genetic programming-based decision trees for software quality classification. In *Proceedings of the 15th IEEE international conference on tools with artificial intelligence (ICTAI'03)*.

Khoshgoftaar, T. M., & Liu, Y. (2007). A multi-objective software quality classification model using genetic programming. *IEEE Transactions on Reliability, 56*(2), 237–245.

Khoshgoftaar, T. M., Liu, Y., & Seliya, N. (2004). Module-order modeling using an evolutionary multi-objective optimization approach. In *Proceedings of the 10th international symposium on software metrics, (METRICS'04)*. Washington, DC, USA: IEEE Computer Society.

Khoshgoftaar, T. M., Liu, Y., & Seliya, N. (2004). A multiobjective module-order model for software quality enhancement. *IEEE Transactions on Evolutionary Computation, 8*(6), 593–608.

Kitchenham, B. A. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-001, UK (July 2007). <http://www.dur.ac.uk/ebse/>.

Kitchenham, B. A., Mendes, E., & Travassos, G. H. (2007). Cross versus within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering, 33*(5), 316–329.

Kotanchek, M., Smits, G., & Kordon, A. (2003). Industrial strength genetic programming. In R. L. Riolo & B. Worzel (Eds.), *Genetic programming theory and practise* (pp. 239–256). Kluwer [Chapter 15].

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

Koza, J. R., & Poli, R. (2005). In E. K. Burke & G. Kendall (Eds.), *Search methodologies – Introductory tutorials in optimization and decision support techniques*. New York, USA: Springer Science and Business Media, Inc. [Chapter 5 – Genetic programming].

Langdon, W., Gustafson, S., & Koza, J. (2009). The genetic programming bibliography, last checked: 13 Feb 2009 (2009). <http://www.cs.bham.ac.uk/wbl/biblio/>.

Lanubile, F., & Visaggio, G. (1997). Evaluating predictive quality models derived from software measures: Lessons learned. *Journal of Systems and Software, 38*(3), 225–234.

Lefley, M., & Shepperd, M. J. (2003). Using genetic programming to improve software effort estimation based on general data sets. In *Genetic and evolutionary computation (GECCO'03). LNCS* (Vol. 2724). Springer-Verlag.

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.

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

Liu, Y., Khoshgoftaar, T., & Yao, J.-F. (2006). Developing an effective validation strategy for genetic programming models based on multiple datasets. In *Proceedings of the 2006 IEEE international conference on information reuse and integration*.

Liu, Y., & Khoshgoftaar, T. M. (2001). Genetic programming model for software quality classification. In *Proceedings of the 6th IEEE international symposium on high-assurance systems engineering (HASE'01)*. Washington, DC, USA: IEEE Computer Society.

Liu, Y., & Khoshgoftaar, T. (2004). Reducing overfitting in genetic programming models for software quality classification. In *Proceedings of the 8th IEEE international symposium on high-assurance systems engineering (HASE'04)*. Washington, DC, USA: IEEE Computer Society.

Low, G. C., & Jeffery, D. R. (1990). Function points in the estimation and evaluation of the software process. *IEEE Transactions on Software Engineering, 16*(1), 64–71.

Mair, C., & Shepperd, M. (2005). The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In *Proceedings of the 4th international symposium on empirical software engineering (ISESE'05)*. Los Alamitos, CA, USA: IEEE Computer Society.

Michalewicz, Z., & Fogel, D. B. (2004). *How to solve it: Modern heuristics* (2nd ed.). Springer Verlag.

Myrtveit, I., & Stensrud, E. (1999). A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE Transactions on Software Engineering, 25*(4), 510–525.

Petticrew, M., & Roberts, H. (2005). *Systematic reviews in the social sciences: A practical guide*. Victoria, Australia: Wiley-Blackwell.

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

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>, (With contributions by J.R. Koza). <http://www.gp-field-guide.org.uk>.

Ratzinger, J., Gall, H., & Pinzger, M. (2007). Quality assessment based on attribute series of software evolution. In *Proceedings of the 14th working conference on reverse engineering (WCRE'07)*.

Reformat, M., Pedrycz, W., & Pizzi, N. J. (2003). Software quality analysis with the use of computational intelligence. *Information and Software Technology, 45*(7), 405–417.

Regolin, E. N., de Souza, G. A., Pozo, A. R. T., & Vergilio, S. R. (2003). Exploring machine learning techniques for software size estimation. In *Proceedings of the international conference of the Chilean Computer Science Society*. Los Alamitos, CA, USA: IEEE Computer Society.

Robinson, G., & McIlroy, P. (1995). Exploring some commercial applications of genetic programming. In *Selected Papers from AISB Workshop on Evolutionary Computing*. London, UK: Springer-Verlag.

Shan, Y., McKay, R. I., Lokan, C. J., & Essam, D. L. (2002). Software project effort estimation using genetic programming. In *Proceedings of the 2002 INTERNATIONAL CONFERENCE ON COMMUNICATIONS, CIRCUITS AND SYstems, Piscataway, NJ, USA*.

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

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

Thelin, T. 2004. Team-based fault content estimation in the software inspection process. In *Proceedings of the 26th international conference on software engineering (ICSE'04)*.

Tsakonas, A., & Dounias, G. (2008). Predicting defects in software using grammar-guided genetic programming. In *Proceedings of the 5th Hellenic conference on artificial intelligence (SETN'08)*. Berlin, Heidelberg: Springer-Verlag.

Wood, A. (1996). Predicting software reliability. *Computer, 29*(11), 69–77.

Zhang, Y., & Yin, J. (2008). Software reliability model by AGP. In *Proceedings of the IEEE international conference on industrial technology, Piscataway, NJ, United States*.

Zhang, Y., & Chen, H. (2006). Predicting for MTBF failure data series of software reliability by genetic programming algorithm. In *Proceedings of the 6th international conference on intelligent systems design and applications (ISDA'06)*. Los Alamitos, CA, USA: IEEE Computer Society.