An Evolutionary Approach to Estimating Software Development Projects[*]

Jesús S. Aguilar–Ruiz [a,*]   Isabel Ramos [a]   José C. Riquelme [a]
Miguel Toro [a]

[a]*Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. Avda. Reina Mercedes s/n. 41012 Sevilla. Spain.*

Abstract

The use of dynamic models and simulation environments in connection with software projects paved the way for tools that allow us to simulate the behaviour of the projects. The main advantage of a Software Project Simulator (SPS) is the possibility of experimenting with different decisions to be taken at no cost. In this paper, we present a new approach based on the combination of an SPS and Evolutionary Computation. The purpose is to provide accurate decision rules in order to help the project manager to take decisions at any time in the development. The SPS generates a database from the software project, which is provided as input to the Evolutionary Algorithm for producing the set of management rules. These rules will help the project manager to keep the project within the cost, quality and duration targets. The set of alternatives within the decision-making framework is therefore reduced to a quality set of decisions.

Key words: Software Development Projects, Software Project Simulators, Machine Learning, Evolutionary Algorithms.

1 Introduction

The use of dynamic models and simulation environments (Stella, Vensim, iThink, Powersim, etc.) in connection with software projects paved the way

for tools that allow us to simulate the behaviour of such projects at the start of the 1990s. With these tools, which are named Software Project Simulators (SPS), the project manager could experiment with different management policies at no cost, in order to take a decision that might be the most suitable and accurate. Questions such as "what would happen if ...?" or "what is happening ...?" or "what would have happened if ...?" are sometimes crucial.

The achievement of the goals of a software project depends on three factors:

- The initial estimations of the necessary resources.
- The management policies to be applied.
- The characteristics of the organization: maturity level, experience, availability of resources, etc.

The dynamic models for software projects have a set of initial parameters that define the management policies to be applied. These policies are associated with the organization (maturity level, average delay, turnover of the project's work force, etc.) and the project (number of tasks, time, cost, number of technicians, software complexity, etc.). The use of an SPS [2] will be complex if the number of parameters is very large. For example, the dynamic model shown in [1] has about 60 parameters. Therefore, the impact on the project of 60 different features could be known. The more parameters the model has, the more complex the development of the project and the use of the model.

If someone asked the project manager "what will be the averaged dedication of the technicians?", the answer would probably be "from 70% to 80%", or something like that. Greater precision is unlikely (for example, an answer like "76%"). The project manager usually adds contingency factors. Machine learning techniques avoid some of these problems by using intervals instead of simple values. In general, these techniques produce decision rules. When decision rules are used as management policies, they are called management rules.

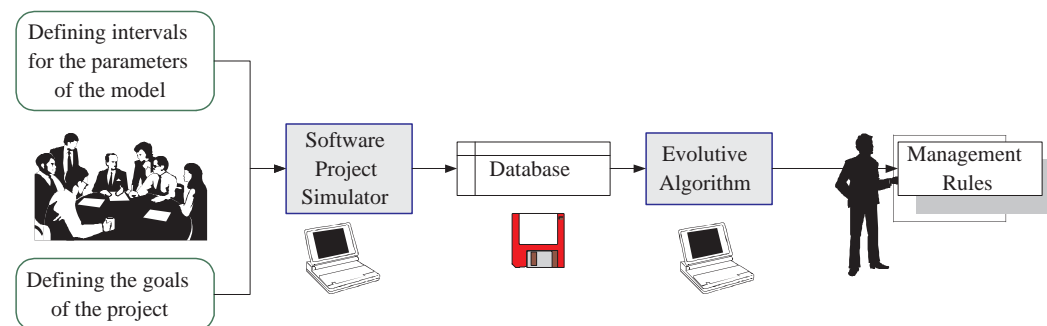Management rules are obtained by following the steps shown in Figure 1.

Fig. 1. Automatic generation of management rules.

Decisions made in any organizational setting are based on what information is actually available to the decision–makers. The computer simulation tools of dynamical systems provide us with the possibility of changing one or several factors while the remaining ones are kept unchanged. The implications of managerial policies on the software development process could be analysed to infer the best decision.

However, a knowledge-based system might help us to determine a subset of decisions more accurately. The simulation of the project produces this knowledge with the database. Then we only have to extract that knowledge in a comprehensible way, for example, decision lists or decision trees. In these data structures we can find an easy-to-understand action to match the actual project results with the project estimations. The search for good decisions from the database generated by simulation is a highly complex task.

Evolutionary algorithms provide us with a method for finding good solutions (in our case, decision rules) in a complex search space (database generated by an SPS) where parameters do not have an obvious relationship.

Other modelling techniques have been applied for estimating effort or cost of software projects: ordinary least-squares regression, analogy-based estimation [6,12,14], genetic programming [4].

2 Evolutionary Algorithm

Evolutionary Algorithms (EA) are a family of computational models inspired by the concept of evolution. These algorithms employ a randomized search method to find solutions to a particular problem [7]. This search is quite different from the other learning methods. An EA is any population-based model that uses selection and recombination operators to generate new sample examples in a search space. The main task in applying EAs to any problem consists in selecting an appropriate representation (coding) and an adequate evaluation function (fitness). In classical EA the members of the population (typically maintaining a constant size) are represented as fixed-length strings of binary digits. The length of the strings and the population size P are completely dependent on the problem. The population simulates the natural behaviour, since the relatively "good" solutions produce offspring which replace the relatively "worse" ones, retaining many of the features of their parents. The estimate of the quality of a solution is based on a fitness function, which determines how good an individual is within the population in each generation. New individuals (offspring) for the next generation are formed by using (normally) two genetic operators: crossover and mutation. Crossover combines the features of two individuals to create several (commonly two) individuals.

Mutation operates by randomly changing several components of a selected individual.

Our system used an EA to search the best solutions and produced a hierarchical set of rules [10]. The hierarchy follows that an example will be classified by the ith-rule if it does not match the conditions of the $(i-1)^{th}$ precedent rules. The rules are sequentially obtained until the space is totally covered. The behaviour is similar to a *decision list* [11]. The structure of the set of rules will be as shown in figure 2.

If conditions Then class
Else If conditions Then class
Else If conditions Then class
....
Else "unknown class"

Fig. 2. Hierarchical set of rules.

As mentioned in [5], one of the primary motivations for using real-coded EAs is the precision to represent attributes values and another is the ability to exploit the gradualness of functions of continuous attributes. For that reason our algorithm uses real coding.

2.1 Coding

In order to apply EAs to a learning problem, we need to select an internal representation of the space to be searched and define an external function that assigns fitness to candidate solutions. Both components are critical for the successful application of the EAs to the problem of interest.

The representation of an individual takes the form shown in Figure 3, where l_i and u_i are values representing an interval for the attribute. The last position (class) is the value for the class. The number of classes determines the set of values to which it belongs, i.e. if there are five classes, the value will belong to the set $0, 1, 2, 3, 4$.

| l_1 | u_1 | l_2 | u_2 | ...... | l_m | u_m | class |

Fig. 3. Representation of an individual of the genetic population.

Each rule will be obtained from this representation, but when $l_i = min(a_i)$ or $u_i = max(a_i)$, where a_i is an attribute, the rule will not have that value. For example, in the first case the rule would be $[-, v]$ and in the second case

$[v, -]$, v being any value within the range of the attribute. If both values are equal to the boundaries, then the rule $[-, -]$ arises for that attribute, which means that it is not relevant. Under these assumptions, some attributes might not appear in the set of rules.

2.2 Algorithm

The algorithm is a typical sequential covering EA [8]. It chooses the best individual of the evolutionary process, transforming it into a rule which is used to eliminate data from the training file [13]. In this way, the training file is reduced for the following iteration. A termination criterion could be reached when more examples to cover do not exist. The method of generating the initial population consists in randomly selecting an example from the training file for each individual of the population. Afterwards, an interval to which the example belongs is obtained by adding and subtracting a small random quantity from the values of the example.

2.3 Fitness function

The fitness function must be able to discriminate between correct and incorrect classifications of examples. Finding an appropriate function is not a trivial task, due to the noisy nature of most databases.

The evolutionary algorithm maximizes the fitness function f for each individual. It is given by the equation 1.

$$f(i) = 2(N - CE(i)) + G(i) + coverage(i) \tag{1}$$

where N is the number of examples being processed; $CE(i)$ is the class error, which is produced when the example i belongs to the region defined by the rule but does not have the same class; $G(i)$ is the number of examples correctly classified by the rule; and the *coverage* of the ith rule is the proportion of the search space covered by such rule. Each rule can be quickly expanded to find more examples thanks to the coverage in the fitness function.

3 Post-mortem analysis of a software project

The project selected for study in this paper is a Personnel Management System project which was carried out jointly by two local software companies. The

Name	Interval	Description	Estimated Value
ADMPPS	(0.3-1)	Average daily manpower per staff (day/day)	0.4
AQADLY	(5-15)	Average delay for quality assurance (days)	5
HIASDY	(5-100)	Average hiring and assimilation delay (days)	20
INUDST	(0.3-1)	Initial understaffing factor (dimensionless)	1
MNHPXS	(1-4)	Most new hires per experienced staff (tech./tech.)	1.5
MXSCDX	(1-1E6)	Maximum schedule completion date extension (dim.)	5
TRNSDY	(1-15)	Time delay to transfer people out (days)	1
DLINCT	(5-15)	Average delay in incorporating discovered tasks (days)	5
AVEMPT	(500-1000)	Average employment time (days)	1000
TRPNHR	(0.1-0.4)	Number of trainers per new employee (dimensionless)	0.15
UNDESM	(0-50)	Man-days underestimation fraction (dimensionless)	48
UNDEST	(0-50)	Tasks underestimation fraction (dimensionless)	15

Table 1

Parameters and variables of the environment of the project and organization. (This notation is the one used in the works of Abdel-Hamid [1].)

data pertain to the design, coding and test phases. Therefore, the analysis phase and other final activities are not considered. The data for initialising the parameters of the SPS were collected from the tracking documents of the original project and the experience of the project manager.

Initially, the effort expended on the project was estimated to require 208 man-days and the project was estimated to be completed in 101 days. The actual figures were 404 man-days and 141 days, respectively. Therefore, the underestimations of effort and time for this project were 48% and 28%, respectively. The average number of technicians involved in the project was 6 and the number of lines of code (LOC) was 67800. The project manager defined a task as 270 LOC.

The real effort spent on the activities of development (design and coding) was 85% ($404 \times 0.85 = 343$), the rest being spent on tests. Assuming that the 10% of the effort expended on development ($343 \times 0.1 = 34$) was used for revision activities, the total effort of development was 309 man-days ($343 - 34 = 309$). Thus, the development productivity for this project was as follows:

$$\text{software development productivity} = \frac{\text{size in LOC}}{\text{development effort}} =$$
$$= \frac{67800}{309} = 219.4 \quad LOC/m - d$$

Some interesting data from the project are shown in Table 1. Each row contains the name of the parameter in the model, the range of values it can assume, a brief description of its meaning and the estimated value at the beginning of the project.

Name	Numeric Interval	Qualitative Description
	(0.3-0.5)	Low
ADMPPS	(0.5-0.75)	Average
	(0.75-1.0)	High
UNDESM	(0-15)	Low
	(15-35)	Average
UNDEST	(35-50)	High
	(0.1-0.15)	Low
TRPNHR	(0.15-0.25)	Average
	(0.25-0.40)	High

Table 2

Qualitative description of the parameters.

Several qualitative labels, which are illustrated in Table 2, were assigned to each parameter, depending on its range of numeric values. This information was provided by the project manager.

3.1 Project simulation

The evolution of the necessary effort, delivery time and pending tasks required in order to complete the Personnel Management System obtained by the SPS is illustrated in Figure 4. If the results are compared with the real values, we can observe that in the simulation the estimated delivery time is 151 days (instead of 141) and the estimated effort is 410 man-days (instead of 404). This means that the absolute percentage error was about 6.62% and 1.46%, respectively. If an SPS had been used at first, we would have known the behaviour of the project, and the absolute percentage error might have been smaller. In addition, in Figure 4 it is possible to note that the first adjustments were made near the middle of the project (around 70 days) when it was detected that the number of pending tasks was greater than that expected. The decision coincided with the real evolution of the Personnel Managment System, since more than half of the estimated time had been spent when more than half of the pending tasks remained still unfinished. Time and final effort might increase if the gradient of the pending tasks does not change. Two additional considerations can be observed in Figure 4: first, the estimated initial effort is modified before the time, as is usual in the majority of the projects being developed by the two local software–development companies; and second, the changes applied to effort and time are made simultaneously.

Data from the real evolution of the project were not collected by the companies. We only know the initial and final data provided by the project manager. Therefore, if both simulated and real behaviours are compared (this was possible, thanks to the help of the project manager), we could only see that the final
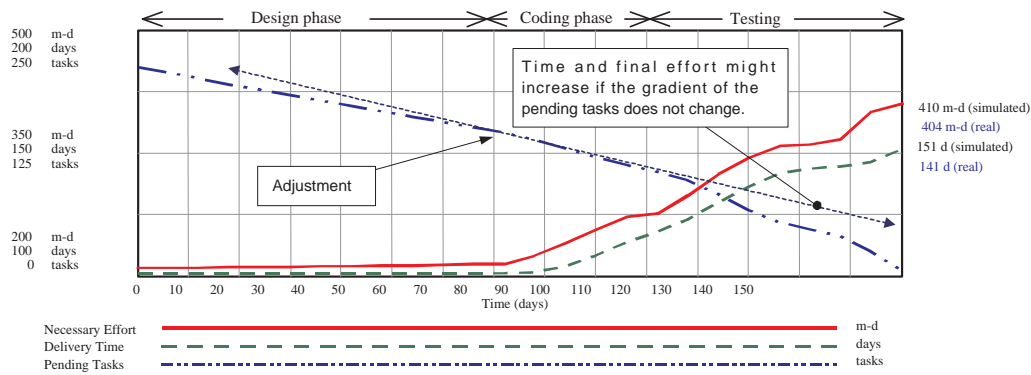
Fig. 4. Evolution of the necessary effort, delivery time and pending tasks for the Personal Managment Software (nominal simulation), obtained by the Software Project Simulator.

estimated time and effort obtained by means of simulation are in agreement with the reality. This is especially true in two respects:

- The early revisions were done when half of the time had been spent.
- The revisions that were carried out to adjust the detected deviations affected both the time and the effort simultaneously.

The utility of an SPS in analysing the evolution of projects carried out by our local software–development companies has been demonstrated, and generally any project could be analysed in the same way [1].

4   Management rules

4.1   The goals of the project

The two goals of the management rules that we are going to obtain are the following:

- We would wish that the values for effort were less than or equal to the value obtained by the simulation (410 m-d). These possible values are labelled as GOOD. The values greater than 410 m-d are labelled as BAD.
- We would wish that the development time were less than or equal to the value obtained by the nominal simulation (151 days). These values are labelled as GOOD. The values greater than 151 are labelled as BAD.

8

Management rules are obtained by the following steps (see Figure 1):

- Define the intervals of values for the parameters of the dynamic model (see Table 1).
- Define the goals of the project (values for time and cost).
- Generate the database automatically: Each simulation produces a record with the values of the parameters and the values of the variables (cost and time) and this record is saved in a file.
- From the file generated in the preceding step, a set of management rules is provided automatically for the decision-making task.

It is important to note that the first two steps aforementioned are performed by the project manager and the next two items are executed automatically.

The management rules obtained by evolutionary algorithms for estimating simultaneously GOOD results for both time and effort are shown in Figure 5. The number of the rule and the number of records from the database covered by the rule are shown respectively, in brackets.

ADMPPS $\geq$ 0.52, TRPNHR $\leq$ 0.38 UNDESM $\geq$ 6, UNDEST $\leq$ 48 INUDST $\leq$ 0.96, HIASDY $\leq$ 32	(Rule 1, 21)
TRPNHR $\leq$ 0.15, UNDESM $\leq$ 37 15 $\leq$ HIASDY $\leq$ 33	(Rule 2, 7)
ADMPPS $\geq$ 0.75, UNDESM $\leq$ 15 INUDST $\geq$ 0.82, 34 $\leq$ HIASDY $\leq$ 45	(Rule 3, 6)
0.55 $\leq$ ADMPPS $\leq$ 0.7, TRPNHR $\leq$ 0.2 UNDEST $\geq$ 14, INUDST $\geq$ 0.47 16 $\leq$ HIASDY $\leq$ 68	(Rule 4, 3)

Fig. 5. Management rules obtained by the evolutionary algorithm for estimating good results for the necessary effort and delivery time simultaneously.

The rules chosen to analyse the project are Rule 1 and Rule 2, since Rule 1 covers more than half the cases forecast as GOOD and Rule 2 has the smaller number of parameters. Another good criterion for selecting the most adequate decision rule is choosing the rule having parameters which are easier to modify and control and, if possible, the rule involving the smallest number of parameters. Once the rule set is obtained, we have to select which parameters do not match the initial values. Observing Table 1, we could rewrite the management–rule set by eliminating the conditions that are matched with the initial values. The new management–rule set is illustrated in Figure 6.

if ADMPPS $\geq$ 0.52 **and** INUDST $\leq$ 0.96	(Rule 1, 21)
if UNDESM $\leq$ 37	(Rule 2, 7)
if ADMPPS $\geq$ 0.75 **and** UNDESM $\leq$ 15 **and** 34 $\leq$ HIASDY $\leq$ 45	(Rule 3, 6)
if 0.55 $\leq$ ADMPPS $\leq$ 0.7	(Rule 4, 3)

Fig. 6. Final hierarchical set of management rules.

For example, in the Rule 1 from Figure 5 the values of the parameters TRP-NHR, UNDESM, UNDEST and HIASDY were initially estimated as belonging to the interval defined by the rule (see Table 1), so that the parameters ADMPPS and INUDST must be the only ones to be modified in order to obtain good results (see Figure 6), since the project is already finished.

Analysing the rules, we could state the following:

- Rule 1: if the average daily manpower per staff (ADMPPS) was greater than 52% and the initial understaffing factor (INUDST) was less than 96%, then the project tended to be towards GOOD.

- Rule 2: if the man-days underestimation fraction (UNDESM) was less than 37%, then the project tended to be towards GOOD (if the error in the initial necessary effort had been less than that estimated; in this project the value of UNDESM was 48%).

Although each rule permits good estimates, the SPS allows us to compare the results of the simulation with those which would be obtained by applying each management rule.

For example, the results obtained by the application of Rule 2 are shown in Figure 7. Both effort and time are remarkably lower than those obtained from the simulation. In concrete terms, the necessary effort and time would be reduced to 26 man-days and 13 days, respectively, if Rule 2 had been applied. With this information, the project manager decides which rule is more appropriate for achieving the aims.

The SPS plays an important role in the decision–making task: first, post-mortem projects can be analysed in order to infer which actions could improve the results; second, an a priori analysis would indicate the intervals within which the values of the parameters have a tendency to achieve the aims of the project.

Taking an effective decision is a very complex task. Some criteria are contradictory. For example, in general, keeping to the project deadline has highest priority, i.e. if the project schedule is at risk, more manpower will be added to the project [9]. However, the Brooks' Law states that adding more people
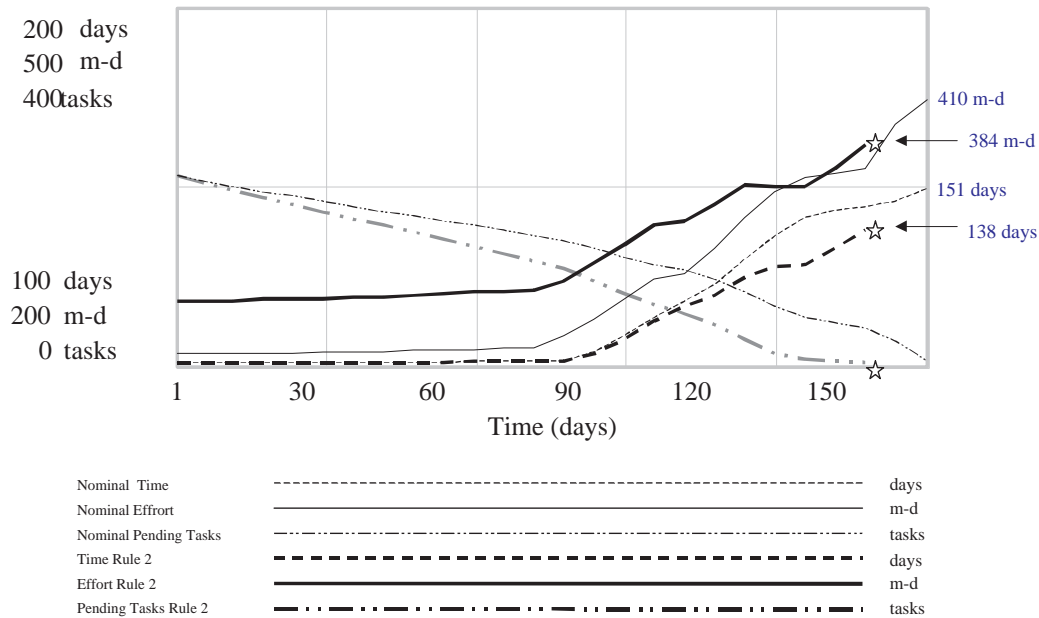
10

Fig. 7. Results obtained by applying Rule 2.

to a late software project makes it later [3].

4.3 Another approach

In the previous section, we saw how an evolutionary algorithm automatically provides management rules from a database generated by an SPS. Another interesting approach consists in presenting graphically the relationship between two parameters in the cases in which good results were obtained. These graphics inform the project manager about the range of values for the parameters being analysed. An example of this kind of analysis is shown in Figure 8, where the parameters TRPNHR and ADMPPS are compared, indicating when the effort and the time are simultaneously good. In Figure 8, each axis is a parameter of the model and each case labelled as GOOD (in the database) is represented by a point (in the figure). In most of the cases in which the necessary effort and the delivery time are simultaneously GOOD, the number of trainers per new employee (TRPNHR) is less than 25% and the average daily manpower per staff (ADMPPS) is greater than 75%.

Logically, this information is not obtained automatically, and we could not know whether the parameters that are represented have an influence on the results.
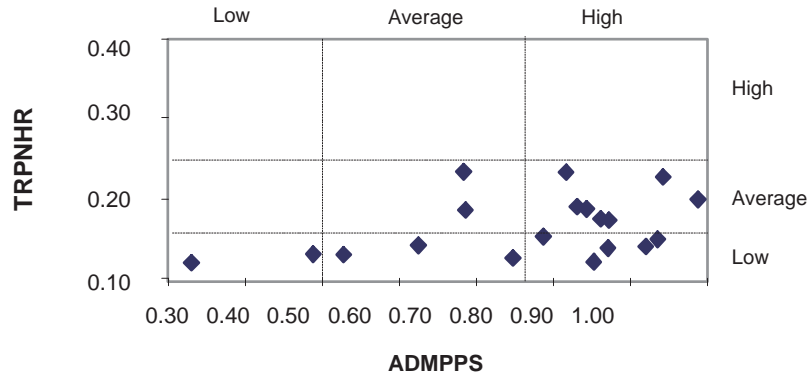
Fig. 8. Relationship between the average daily manpower per staff and the number of trainers per new employee.

5 Conclusions

The use of simulators and systems that learn decision rules helps to estimate software projects and to produce management rules automatically for the decision–making task. These management rules could be applied:

- Before the project has begun: defining more adequate managerial policies.
- After the project has finished: doing a post-mortem analysis.
- When the project is running: taking immediate decisions (monitoring).

Management rules make it possible to:

- obtain values considered as good (very good, normal, bad, etc.) for any variable of interest (time, effort, quality, number of technicians, etc.), independently or together with other variables.
- analyse managerial policies capable of achieving the aims of the project.
- know to which range of values the parameters must belong in order to obtain good results.

In short, it is possible to generate management rules automatically for any software project and to know the managerial policies that ensure the achievement of the initial aims. The deviations from the initial forecast could be detected (monitoring) and the behaviour of the process is well understood through the management–rule set.

Evolutionary Computation provides an interesting approach for dealing with the problem of extracting knowledge from databases generated by Software Project Simulators. The results demonstrate that evolutionary algorithms find good solutions in the search space. And, what is more important, the results (management rules) are applicable and beneficial.

12

The authors consider that this technique contributes to coping with the complex problem of decision-making within the Software Project Development framework, and it facilitates the use of dynamic models, since the project manager only has to provide the aims of the project and the range of the parameters, especially for those having a high degree of uncertainty for him.

We are currently working on the application of other machine learning techniques (fuzzy logic, association rules, etc.) to the databases generated by the SPS.

Acknowledgements

References

[1] T. K. Abdel-Hamid. *Software Project Dynamics: an integrated approach.* Prentice-Hall, 1991.

[2] J. Aroba, I. Ramos, and J. C. Riquelme. Decision making in software project using fuzzy clustering algorithms. In *Proceedings of the Software Process Simulation Modeling Workshop, PROSIM2000*, 2000.

[3] F. P. Brooks. *The Mythical Man Month.* Addison-Wesley, Reading, MA, 1978.

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

[5] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. *Foundations of Genetic Algorithms-2*, pages 187–202, 1993.

[6] G. R. Finnie, G. E. Wittig, and J.-M. Desharnais. A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software*, 39(3):281–289, 2000.

[7] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[8] T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[9] D. Pfahl and K. Lebsanft. Using simulation to analyse the impact of software requirement volatility on project performance. *Information and Software Technology*, 42:1001–1008, 2000.

[10] J.C. Riquelme, J.S. Aguilar–Ruiz, and M.Toro. Discovering hierarchical decision rules with evolutive algorithms in supervised learning. *International Journal of Computers, Systems and Signals*, 1(1):73–84, 2000.

[11] R. L. Rivest. Learning decision lists. *Machine Learning*, 1(2):229–246, 1987.

[12] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12):736–743, 2000.

[13] G. Venturini. Sia: a supervised inductive algorithm with genetic search for learning attributes based concepts. In *Proceedings of European Conference on Machine Learning*, pages 281–296, 1993.

[14] F. Walkerden and R. Jeffery. An empirical study of analogy-based software effort estimation. *Empirical Software Engineering*, 42:135–158, 1999.