Generation of Management Rules through System Dynamics and Evolutionary Computation

Jesús S. Aguilar-Ruiz[1], José C. Riquelme[1], Daniel Rodríguez[2], and Isabel Ramos[1]

[1]Department of Computer Science, University of Seville
41012 Sevilla, Spain
{aguilar,riquelme,Isabel.ramos}@lsi.us.es
[2]Department of Computer Science, The University of Reading
Reading, RG6 6AY, UK
d.rodriguez-garcia@rdg.ac.uk

Abstract. Decision making has been traditionally based on a managers experience. This paper, however, discusses how a software project simulator based on System Dynamics and Evolutionary Computation can be combined to obtain management rules. The purpose is to provide accurate decision rules to help project managers to make decisions at any time in the software development life cycle. To do so, a database from which management rules are generated is obtained using a software project simulator based on system dynamics. We then find approximate optimal management rules using an evolutionary algorithm which implements a novel method for encoding the individuals, i.e., management rules to be searched by the algorithm. The resulting management rules of our method are also compared with the ones obtained by another algorithm called C4.5. Results show that our evolutionary approach produces better management decision rules regarding quality and understandability.

1 Introduction

Decision making is an important part of software processes. Most organisations allocate resources based on predictions and improving their accuracy of such predictions reduce costs and make use of resources in a more effective way.

In this work we have combined System Dynamics and Evolutionary Computation to achieve high quality decision rules. On the one hand, dynamic models such as System Dynamics are becoming popular among the Software Engineering research community as they may provide a better solution to some of the problems found in Software Engineering when compared with traditional static models. In principle, dynamic models can help in making good decisions with data that are scarce and incomplete. On the other hand, Evolutionary algorithms allow us to find an approximate optimal solutions in a large search space, i.e., management rules in a database generated by the project simulation based on System Dynamics.

The organisation of the paper is as follows. Section 2 discusses the related work. Section 3 presents the approach taken for generating management rules. Section 4 discusses a series of scenarios to prove our approach. Section 5 discusses its validation with a generic framework. Finally, Section 6 concludes the paper and future work is outlined.

2 Related Work

2.1 System Dynamics and Software Project Simulators

System Dynamics is a method for studying how complex systems change over time where internal feedback loops within the structure of the system influence the entire system behaviour [9]. In System Dynamics, it is necessary first to understand the behaviour of the real word so that important cause-effect relationships, which are called base mechanisms, can be modelled. The formal model consist of a set of mathematical equations which can be represented graphically as flow graphs.

The application of system dynamics to software projects provides the perspective of considering them as complex socio-technological systems. Their evolution will be determined by their internal structure as well as the relations established inside the working team. This allows the development of dynamic models to describe the feedback structure of the system being modelled as well as the mental process followed by project managers in making decisions.

The building of a dynamic model for a software project is based on the evolution of the project and the attainment of the project goals such as meeting the deadlines, a project phase being within budget, etc. depends on (i) a set of initial parameters that define the initial estimations and (ii) the management policies to be applied. These policies are related with the project (number of tasks, time, cost, number of technicians, software complexity, etc.) and the organization (maturity level, average delay, turnover on the project's work force, etc). Using of software project simulation environments based on dynamic models such as Vensim, iThink, etc., project managers can experiment different management policies without additional cost including the following:

1. A priori, project analysis, to simulate the project before initiation.
2. Project monitoring, to simulate the project during its development phase for adapting the project estimation to its actual evolution.
3. Post-mortem analysis through the simulation of a finished project, to know how the results obtained could have been improved.

2.2 Abdel-Hamid's System Dynamic Model

Abdel-Hamid [1] developed an model in order to study the effects of project management policies and actions on software development. Developed to help

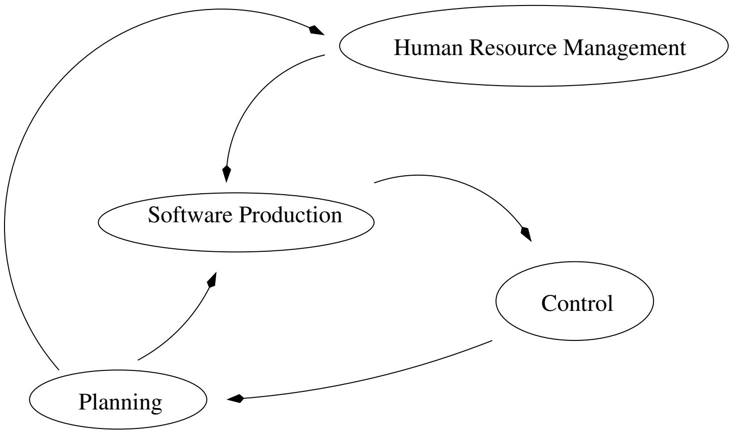

Fig. 1. High Level View of Abdel-Hamid Model of Software Project Management

understanding the software development process, it allows to evaluate the impact of management policies. Being the software development of complex systems difficult to understand in its entirety, Abdel-Hamid's model is partitioned into four subsystems that are manageable and understandable (see Figure 1):

1. Human Resource Management Subsystem deals with hiring, training, assimilation and transfer of the human resources.
2. Software Production Subsystem models the software development process excluding requirements, operation and maintenance. This subsystem has four sectors: Manpower Allocation, Software Development, Quality Assurance and Rework, and System Testing.
3. Control Subsystem deals with the information that decision makers have available. The model takes into account that it is difficult to know the true-state of a process during development as usually such information is inaccurate.
4. Planning Subsystem takes into account the initial project estimates. Such estimates need to be revised through the software project life cycle

2.3 Evolutionary Algorithms

Evolutionary Algorithms (EA) are a family of computational models inspired by the concept of evolution and natural selection. These algorithms employ a randomised search method to find approximate optimal solutions to a particular problem [12]. Generally, this approach is applied to search spaces that are too large to use exhaustive techniques.

Based on a generally constant size population of individuals, an EA follows an iterative procedure based on selection and recombination operators to generate new individuals in the search space. Such individuals are usually represented by a finite string of symbols called chromosome. They encode a possible solution in a given problem search space which comprises of all possible solutions to the problem. The length of the string and the population size are completely dependent of the problem in hand and the finite string of symbol alphabet can be binary, real-valued encodings, tree representations, etc. The population simulates natural evolution in the sense that iteratively good solutions (individuals) generate other solutions (offsprings) to replace bad ones retaining many features of their parents. Therefore, a critical factor is to know how good is a solution which depends on a fitness function, so that high-fitness individuals stand a better chance of reproducing, while others are likely to disappear. Another critical factor is how new solutions are formed. This is usually carried out using two genetic operators named crossover and mutation. Crossover creates a new individual combining parts of its parents representation. Mutation changes randomly a small part of the string that represents the individual.

2.4 Decision Trees and Management Rules

A decision tree is a classifier with the structure of a tree, where each node is a leaf indicating a class, or an internal decision node that specifies some test to be carried out on a single attribute value. A well-known tool for producing decision trees is C4.5 [14] which basically consists of a recursive algorithm with a divide and conquer technique that optimises a decision tree. The main advantage of decision trees is their immediate conversion to rules (in the form of *if ... then ...*) that can be easily interpreted by decision-makers. For example, assume a hypothetical database that associates weight (in kilograms) and height (in meters) of a person with having a paper accepted in a relevant conference. The database is a sequence of tuples such as (83, 1.71, no), (71, 1.62, yes), etc. A rule describing the relationship among attribute values and class might be: *if weight in [60, 70] and height in [1.60, 1.68] then he/she is a candidate for having a paper accepted*. It is to note that (i) the finding of these rules can be tackled with many different techniques, one such technique being evolutionary algorithms and (ii) such rules may produce relationships between variables that are not be evident.

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

- obtain values considered as *good* 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.

3 Our Approach

Our goal is to find management policies that produce good results for variables of interest in every scenario taking into account that quality must be prioritised, i.e., we must consider only high quality projects. Based on these constraints, we try to obtain policies that minimize the delivery time (independently of the effort), the effort (without taking into account the delivery time) or both delivery time and effort at once. Figure 2 shows the process to obtain decision rules:

1. Define the intervals of values for the attributes of the dynamical model (see Table 2 which will be discussed in the case studies).
2. Define the goals of the project (values for time and cost).
3. Generate the database: each simulation produces a record with the values of the parameters and the values of the variables as it is shown in Table 3 in the next Section.
4. Assign labels to the records according to a threshold for every variable.
5. Generate decision rules from the previously generated database using our evolutionary algorithm.
6. Compare generated decision rules with those produced by using the C4.5 algorithm.

It is to note that the first two steps previously are performed by the project manager and the next two items are executed automatically (see Figure 2).

3.1 System Dynamics

In our approach, decision rules are generated by the evolutionary algorithm based on the database generated by the Abdel-Hamid's dynamic model [1] simulating the temporal behaviour of the project. It contains the controls for setting three groups of required parameters: the parameters of the project (size, number technicians etc.), the parameters of the organisation (average delays related to hiring, dismissals etc.) and finally, the parameters controlling the simulation process. Our generated database consist of a sequence of tuples with continuous attributes and a class label.

Abdel-Hamid’s work was chosen because in addition to emphasise project management, descriptions, validation and conclusions of his model are extensively reported.

3.2 Evolutionary Computation

Evolutionary computation provides an interesting approach for dealing with the problem of extracting knowledge from databases. In this case, the search space comprises of management rules for decision making and we will try to find a rule set

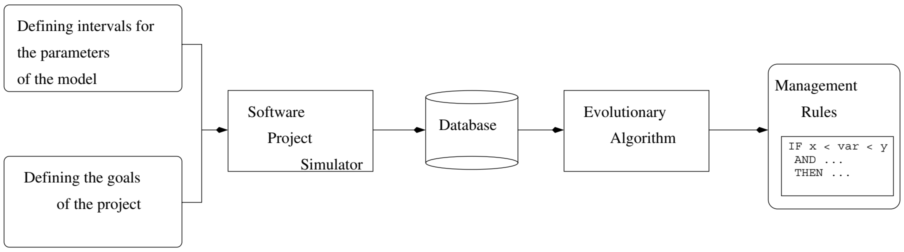
Defining intervals for the parameters of the model
Defining the goals of the project
Software Project Simulator
Database
Evolutionary Algorithm
Management Rules
IF x < var < y
AND ...
THEN ...

Fig. 2. Generation of Rules

that describes the knowledge within data in order to classify it. For an attribute a_i, the rules take the form of „if a_j in $[l_j, u_j]$ then *class*", where l_j and u_j are two real values belonging to the range of the attribute and $l_j < u_j$.

Two critical factors influence the decision rules obtained by using evolutionary algorithms: (i) the selection of an internal representation of the search space (coding) and (ii) the definition of an external function that assigns a value of goodness to the potential solutions (fitness).

3.2.1 Coding

Our approach, called „natural coding", uses one gene per continuous attribute leading to a reduction in the search space, i.e., the space of all feasible solutions. This reduction has a great influence on the convergence of the evolutionary algorithm, i.e., finding approximate optimal rules. Our coding method uses natural numbers so that a value per attribute is necessary, i.e., every interval is encoded by one natural number, leading to a reduction of the search space size. Natural Coding is formally described in [2]. The reader can have a better understanding of this coding method, for continuous as well as discrete attributes, by referring to that work.

3.2.2 Fitness Function

The fitness function must be able to discriminate between correct and incorrect classifications of samples. 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 and it is given by the equation (1).

$$f(i)=2(N-CE(i))+G(i)+coverage(i) \qquad (1)$$

where N is the number of examples being processed; *CE(i)* is the number of class errors, which is produced when an example belongs to the region defined by the rule but it does not have the same class; *G(i)* is the number of examples correctly classified by the rule; and the *coverage* of the *i-th* rule is the proportion of the search space covered by such rule.

3.2.3 Algorithm

The algorithm is a typical sequential covering EA [13], i.e., it chooses the best individual of the evolutionary process, transforming it into a rule which is used to eliminate data from the training file [17]. In this way, the training file is reduced in the next iteration. The method of generating the initial population consists of randomly selecting an example (with the label of interest) 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. A termination criterion is reached when there is no more examples to cover.

Table 1 gives the values of the parameters involved in the evolutionary process. It is worth noting that the decision rules presented below were obtained by running the evolutionary algorithm 50 generations. This means that the running time of the algorithm was very small (less than a minute in a Pentium II 450MHz).

Table 1. Parameters of the evolutionary algorithm

Parameter	Value
Population Size	100
Generations	50
Crossover probability	0.5
Individual Mutation probability	0.1
Gene mutation probability	0.1

4 Case Studies: Generation of Rules for Software Development Projects

By simulating the Abdel-Hamid's dynamic model [1] we obtained three different scenarios. These scenarios are defined by intervals of values that the attributes can take. It is worth noting that the initial values were almost the same as those described by Abdel-Hamid. Only four attributes were modified, taking random values from previously defined intervals. Each row comprises of the name of the attribute in the model, the range of values it can take, a brief description of its meaning and its estimated value at the beginning of the project.

The attributes ASIMDY, HIREDY, TRNSDY (related with the new personnel management enrolled to the ongoing project) and MXSCDX (related with the decision of imposing constraints to the delivery time) described in Table 2, will allow us to analyse their influence on the variables of the project (mainly, delivery time, cost or effort, and quality) described in Table 3. Table 2 shows attribute names and their description, range and initial estimated input value. Table 3 shows attribute names, description and initial estimated output value. These input and output parameters will be used to generate the following scenarios:

– Scenario 1: Attributes can take any value within the range (as defined in Table 2).
– Scenario 2: Attributes related with the personnel management take the low values in their range, i.e., ASIMDY in [5,15], HIREDY in [5,10] and TRNSDY in [5,10], so that the personnel management is fast. Moreover, the date extension can not be greater than 20% of the initially estimated value.
– Scenario 3: Personnel management is the same as in the previous scenario and MXSCDX can take any value within its range.

Table 2. Parameters of the environment of the project and organisation

Attribute	Interval	Description	Estimated Value
ASIMDY	[5,120]	Average assimilation delay (days)	20
HIREDY	[5,40]	Average hiring delay (days)	30
TRNSDY	[5-15]	Time delay to transfer people out (days)	10
MXSCDX	[1-50]	Max schl completion date (dimimensionless)	3

Table 3. Variables of the environment of the project and organisation

Variable	Description	Estimated Value
EFFORT	Effort for the project development (tech. per day)	1.111
TIME	Delivery time (days)	320
QUALITY	Quality of the final product (errors/task)	0

Table 4. Number of Software Development Projects satisfying the constraints. Rows indicate constraints over variables and columns over attributes: (1) None; (2) EFFORT < 1888; (3) TIME < 384

Constraint	(1)	(2)	(3)
NONE	300	42	37
QUALITY < 0.35	8	1	8
QUALITY < 0.45	227	26	27

4.1 Scenario 1

The results for this scenario are as follows: the variable EFFORT take values within the range [1589,3241], TIME in [349.5,437] and QUALITY in [0.297,0.556]. In order to assign a label to the records, the variables were discretised, taking one threshold for TIME and EFFORT and two for QUALITY. The constraints over the attributes and variables are shown in Table 4, where rows indicate constraints over variables and columns over attributes. Only 8 out of three hundred simulations had QUALITY less than 0.35, whereas when QUALITY is less than 0.45 there were 26 records with EFFORT under 1888 and 27 records with TIME under 384. Only one record satisfied both constrains EFFORT and TIME simultaneously.

We used the evolutionary algorithm to find rules that provide an adequate effort, a good delivery time or an excellent quality (three independent analysis).

Rule for effort: a rule covering 22 records (C4.5 produces two rules covering 23 records).

```
if 101≤ ASIMDY
   and 20.2≤ HIREDY
   and 6.9≤ MXSCDX
     then EFFORT≤ 1888 and QUALITY≤ 0.45
```

If the assimilation and the hiring are slow then the effort is optimised.

Rule for time: a rule covering 20 records (C4.5 produces two rules covering 21 records).

```
if 7≤ ASIMDY≤ 17.9
   and 5.6≤ HIREDY≤ 39.5
   and 2.3≤ MXSCDX
   and 5.4≤ TRNSDY
     then TIME≤ 384 and QUALITY≤ 0.45
```

This rule shows that the intervals for the attributes HIREDY, MXSCDX and TRNSDY are very unrestricted, i.e. these attributes might take any value within the range. Therefore, the rule could only consider the attribute ASIMDY: a fast assimilation should enough to fulfil the time constrains.

Rule for quality: if QUALITY must be less than 0.35, only 8 records with TIME less than 384 were obtained. The evolutionary algorithm needs one rule (while C4.5 needs two rules).

```
if ASIMDY≤ 23.2
   and 5.9≤ HIREDY≤ 32.8
   and 13.1≤ MXSCDX≤ 47.1
   and 5.5≤ TRNSDY≤ 10.8
     then TIME≤ 384 and QUALITY≤ 0.35
```

This rule is similar to the previous one but there is a warning about minimising the average delay to transfer people.

4.2 Scenario 2

For this scenario the variable EFFORT takes values in [1709,3395], TIME in[349.5, 354.3] and QUALITY in [0.235, 0.661]. The thresholds for labelling the database were 1999 for EFFORT, 352 for TIME and 0.35 and 0.45 for QUALITY. The number of records satisfying these constraints are shown in Table 5.

Rules for time: two rules covering 31 records out of 45. (C4.5 produces 3 rules covering 22 records).

```
if 11.6 ≤ ASIMDY
   and 6.3 ≤ HIREDY
   and 1.1≤ MXSCDX ≤ 1.15
     then TIME≤ 352 and QUALITY≤ 0.45

if 8.6 ≤ ASIMDY
   and 6.8 ≤ HIREDY ≤ 9.5
   and 1.16 ≤ MXSCDX
   then TIME ≤ 352 and QUALITY ≤ 0.45
```

These rules are complementary in relation to the schedule completion date extension: the first one has a fixed schedule and the second one has not.

Table 5. Number of Software Development Projects satisfying the constraints. (1) None; (2) EFFORT < 1999; (3) TIME < 352

Constraint	(1)	(2)	(3)
NONE	300	105	297
QUALITY < 0.35	13	0	12
QUALITY < 0.45	47	1	45

Rules for quality: for 12 records with QUALITY less than or equal to 0.35 the evolutionary algorithm produced a rule covering six of them (C4.5 produced three rules covering 8 out of 12):

```
if 11.8 ≤ ASIMDY
   and 7.8 ≤ HIREDY ≤ 9.8
   and 1.1 ≤ MXSCDX ≤ 1.19
   and TRNSDY ≤ 6.8
     then TIME ≤ 352 and QUALITY ≤ 0.35
```

To improve the quality, the rule must limit the time delay to transfer people out.

4.3 Scenario 3

The model is again simulated 300 times with the following values for the variables: EFFORT in [1693,2415], TIME in [349.5,361.5] and QUALITY in [0.236,0.567]. The threshold for labelling the database are the same that in the second scenario. The number of records satisfying the constraints is shown in Table 6.

Rules for effort and time: the scenario 3 is the only one that has a case which both constraints over time and effort were satisfied: when EFFORT < 1999 and QUALITY <0.45 there are 11 cases matching TIME <352 by chance. Two rules cover these records (C4.5 needs three rules).

```
if 8.4 ≤ ASIMDY ≤ 11.5
   and 8.8 ≤ HIREDY
   and 9.3 ≤ MXSCDX
   then EFFORT ≤ 1999 and TIME ≤ 352
         and QUALITY ≤ 0.45 (7 records)

if 9.8 ≤ ASIMDY ≤ 11.2
   and 6.8 ≤ HIREDY ≤ 8
   and MXSCDX ≤ 39.5
     then  EFFORT ≤ 1999 and TIME ≤ 352
        and QUALITY ≤ 0.45 (4 records)

  if 9.8 ≤ ASIMDY ≤ 11.2
     and 6.8 ≤ HIREDY ≤ 8
     and MXSCDX ≤ 39.5
       then  EFFORT ≤ 1999 and TIME ≤ 352
           and QUALITY ≤ 0.45 (4 records)
```

Table 6. Number of Software Development Projects satisfying the constraints (1) None; (2) EFFORT < 1999; (3) TIME < 352

Constraint	(1)	(2)	(3)
NONE	300	164	226
QUALITY < 0.35	43	0	9
QUALITY < 0.45	116	11	49

These rules point out that for obtaining good results simultaneously (TIME) and EFFORT) is essential a fast assimilation of technicians. The two rules are complementary in relation to HIREDY and MXSCDX.

Rules for time: if we only wish to minimise the variable TIME, we can produce similar rules as before by relaxing the assimilation (ASIMDY). In this way, we can accept values for this parameter less than 14.

5 Validation and Discussion

Kitchenham *et al.*[11] propose a framework for validating a bidding system, which can also be used for evaluating Dynamic Systems. The evaluation framework is composed of five quality aspects: (i) syntactic quality, (ii) semantic quality, (iii) pragmatic quality, (iv) test quality and (v) value quality. For each quality aspect, the authors define goals and means to achieve them. We will use this framework for discussing our approach.

The syntactic quality goal is syntactic correctness. Systems Dynamics have a well defined syntax based on graphs and formal models so that they are amenable to automation. Rules and decision trees have also a well defined syntax.

The semantic quality goal comprises of feasible validity and feasible completeness. In the context of System Dynamics, feasible completeness mean that the model include all the relevant causality relationships in the domain. Feasible validity refers to the correctness of such relationships to the domain. Kitchenham *et al.* define 'traceability to the domain' as the model property that supports the semantic quality goals.

As we have stated previously, Abdel-Hamid's is not only well documented, but it was also semantically validated in several ways: (i) replication of reference models based on extensive review of the literature, (ii) face validity based on interviews with software project managers at major organizations to fill in gaps in the literature, (iii) extreme conditions simulation, (iv) case study parameterising it for a NASA software development project and reproducing the patterns of the completed software project (cost, schedule, and work force loading).

In our opinion, semantic quality goals are the most difficult to achieve in the context of System Dynamics and since they are related to the problem of knowledge elicitation. Another important point highlighted by Kitchenham *et al.* is the difference between generic and specialized models. The same applies to System Dynamics, where causal relationships of software processes could be defined in a generic way but once these generic processes are modelled it can be difficult to adapt them to other more concrete environments.

The pragmatic quality deals with the issue of how to express a model and it comprises of two goals: feasible comprehension and feasible understandability. Methods to facilitate comprehension include visualization, explanation and filtering which can be assessed through empirical studies. Expressive economy and structuredness are model properties that enable the feasible understandability quality aspect. In our case, the documentation and guidelines provided by Abdel-Hamid help to customise and improve the model. In relation to the set of rules are intuitive and self-documented.

Test Quality The goal for the test quality aspect is feasible test coverage. Simulation studies can be used to assess this quality aspect.

Value Quality refers to practical utility. Kitchenham *et al.* include appropriate user manuals, training etc. as means to achieve it. In the context of System Dynamics, project managers and quality assurance engineers need to understand the underlying models to apply them correctly. In relation to the quality of the set of rules produced by the evolutionary algorithm, there are many advantages of our approach in comparison with C4.5 [14]. For example, our algorithm searches for rules which cover records with the label identified by the user whereas C4.5 generates a decision tree in which all the labels appear, i.e., including those not helping to manage the project properly; from these rules we cannot extract useful knowledge related to management policies. C4.5 always produced more rules covering less records. In many cases, the intervals found by C4.5 for some parameters had a very small range, which is inappropriate if project manager want to vary such parameters to achieve a goal.

Finally, we must comment that System Dynamics models may not be applied in all software organisations. There are constrains that need to be born in mind. First, expertise is required to model the entities, attributes and cause-effect relationships that compose such models. Second, models need to be customised, i.e., calibrated for each organization. To do so, organizations may need a well established process for applying System Dynamic properly. If we consider the Capability Maturity Model (CMM) [15], at level 3, Key Process Areas (KPA) related to managerial issues and organization-wide standards are introduced. At this level, data collection is integrated in the development process. Moreover, it is only at Level 4 that an organisational measurement program is established for measuring the quality of the products and processes. As a result, availability of data in order to calibrate the System Dynamic model is usually only available in mature organisations.

6 Conclusions and Future Work

In this paper, we have shown an approach that combines system dynamics and evolutionary computation to produce rules automatically for the decision-making task in project management. Our technique may contribute 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. Such management rules can be applied before, during and after the execution of a project.

In our approach, we first generate a database using the Abdel-Hamid's model [1] simulating the temporal behaviour of the project. Then, decision rules are constructed by the evolutionary algorithm, which is based on a new method for coding. Such rules were compared with the ones obtained by another well-known algorithm (C4.5[14]). It produces better solutions in the search space, covering more *good* examples with less number of rules being the results more applicable and beneficial.

In addition to further improvement such evolutionary approaches, we are currently working on the application of other machine learning techniques (fuzzy logic, association rules, etc.) to the databases generated by the simulation. From the software engineering point of view, our future work is related to how to integrate such approach into the development process and visualisation. We have detected as a basic prerequisite the lack of a methodology for interpreting all the information that project managers collect from multiple sources. Usually, project management tools and control panels inform the project manager about the state of the project. Corrective actions are taken on the basis of the manager's experience and assessed on the basis of management rules, in order to fulfil the desired goals. Further efforts are also directed towards improving the visualization of such rules including all kind of data generated by processes and products.

Acknowledgements

The research was supported by the Spanish Research Agency (CICYT-TIC1143-C03-02 and CICYT- TIC1143-C03-01)

References

1. Abdel-Hamid, T.K.: Software Project Dynamics: an integrated approach. Prentice-Hall, 1991.
2. Aguilar-Ruiz J.S., Riquelme J.C., and Del Valle C.: Improving the Evolutionary Coding for Machine Learning Tasks. *15th European Conference on Artificial Intelligence*, ECAI'02, IOS Press, 173-177, 2002.
3. Aguilar-Ruiz J.S., Riquelme J.C., Ramos I. and Toro M.: An evolutionary approach to estimating software development projects. *Information and Software Technology*, 14(43):875-882, 2001
4. van Leeuwen, J. (ed.): Computer Science Today. Recent Trends and Developments. Lecture Notes in Computer Science, Vol. 1000. Springer-Verlag, Berlin Heidelberg New York (1995)
5. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edn. Springer-Verlag, Berlin Heidelberg New York (1996)
6. Dolado, J.J.: On the problem of the software cost function. *Information and Software Technology*, 43:61-72, 2001.
7. Fayyad, U.M. and Irani, K.B.: Multi-interval discretisation of continuous valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1993.

8. Finnie, G.R., Wittig, G.E., and Desharnais, J.-M.: 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.
9. Forrester, J.W.: *Industrial Dynamics*, Productivity Press, 1961.
10. Holte, R.C.: Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11:63-91, 1993.
11. Kitchenham B.A., Pickard L.M., Linkman S.G. and Jones, P.: A Framework for Evaluating a Software Bidding Model, *Empirical Assessment in Software Engineering (EASE)*, Keele University, 2002.
12. Koza, J.R.: *Genetic Programming*. The MIT Press, Cambridge, Massachusetts, 1992.
13. Mitchell, T.: *Machine Learning*. McGraw Hill, 1997.
14. Quinlan, J.R.: *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, California, 1993.
15. Paul, M.C., Curtis, B., Chrissis, M.B. and Weber, C.V.: Capability Maturity Model, Version 1.1, *IEEE Software*, 10:18-27, 1993.
16. Shepperd, M. and Schofield, C.: Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12):736-743, 2000.
17. Venturini, G.: Sia: A supervised inductive algorithm with genetic search for learning attributes based concepts. In *Proceedings of European Conference on Machine Learning*, pp. 281-296, 1993.
18. Walkerden, F. and Jeffery, R.: An empirical study of analogy-based software effort estimation. *Empirical Software Engineering*, 42:135-158, 1999.