Building Knowledge Discovery-Driven Models for Decision Support in Project Management

María N. Moreno García , Luis A. Miguel Quintales, Francisco J. García Peñalvo y M. José Polo Martín

Dept. Informática y Automática. University of Salamanca. Plaza Merced s/n. 37007 Salamanca. Spain
{mmg, lamq, fgarcia, mjpolo}@usal.es

Abstract. Accurate estimations of software size in the early stages of a software project are critical in software project management because they lead to a good planning and reduce project costs. In this work, the relation between early software size measures as the function points and measures of the final product as the lines of code has been studied. A process to refine association rules, based on the generation of unexpected patterns, is proposed. The goal is to generate strong association rules between attributes that can be obtained early in the project and the final software size. Our approach allows to extract more relevant patterns from data for the effective decision support in the project management area.

1 Introduction

A current trend in software organizations is to take advantage of data-mining techniques to extract knowledge from the huge volumes of data that they manage. Data-mining techniques provides the objective information needed to make decisions about business and engineering performance. In the project management environment many of those decisions are based on the software size estimation.

The determination of the value of software size in the early stages of a software project is a key activity in order to predict the effort to be consumed in a software project. Accurate estimations are critical in software project management because they lead to a good planning and reduce project costs. Many methods for estimating software size have been developed. Most of them provide functional measures of the software size such as functions points [1], functions blocks [2] and object points [3]. The values obtained for those functional variables should correspond with the final product size (for example lines of code). In an earlier work [4] we used several supervised data mining techniques to obtain software size estimation models. A component-based method [5] and a global method (Mark II) [6] were taken as reference. We observed that models obtained by combining attributes from both methods were better than models obtained by using attributes of each method separately. We also found a correlation between the function points of the method Mark II (MKII FP) and lines of code (LOC). This indicates that the function points of the method Mark II are good size predictors, but that correlation is stronger for some size intervals than for others.

In this work we have used unsupervised data-mining algorithms in order to discover knowledge that enhances the results of the classification models obtained previously and lead to the development of more efficient decision-support systems. We have applied two discovery-driven techniques: clustering and association rules. The goal in *knowledge discovery* modeling is to discover rules and segments of the data that behave similarly.

At first, we applied several data clustering algorithms to obtain a high-level view of what is occurring in the data with respect to software size and to classify records. We also applied association rule algorithms to discover pattern in data. Extracted rules revealed which software size intervals showed a weak correlation between lines of code and Mark II function Points. We applied a refinement process in order to obtain stronger rules that reinforce the relation between LOC and MKII FP by using other attributes that can be obtained early in the project life cycle.

Our proposal should provide managers with more relevant patterns from data and aid them in effective decision making.

2 Data Description

This section describes the data set used to build the models. It was obtained from experiments carried out by Dolado [7]. The data originate from academic projects in which students developed accounting information systems that have the characteristics of commercial products. Each system includes some or all of the following subsystems: sales, purchases, inventories, financial statements, and production cycles. We have analyzed data from 42 projects written in Informix-4GL.

The information available was divided in two groups. One group containing global data from each project (42 records) and other group containing attributes from each component of the projects (1537 records). The code of the applications was classified into three types of components according to Verner and Tate [5]

Component attributes

TYPECOMP: type of component (1: menu, 2: input, 3: report/query)
OPTMENU: number of choices (only for menus)
DATAELEMENT: number of data elements (only for inputs and reports/queries)
RELATION: number of relations (only for inputs and reports/queries)
LOC: lines of code

Project attributes

The attributes described below are used in Mark II method [6] to calculate functions points. That proposal consists of considering a system composed of logical transactions. A logical transaction is a unique input/process/output combination triggered by a unique event of interest to the user or a need to retrieve information. The central concept in this method is that of entity, that replaces the concept of logical file.

LOC: lines of code
NOC: number of components
NTRNSMKII: number of transactions MKII
INPTMKII: number of total inputs
ENTMKII: number of referenced entities
OUTMKII: number of total ouputs (data elements over all transactions)
UFPMKII: number of unadjusted functions points MKII

We observed in previous research [4] that classification models obtained by combining attributes from both methods were better than models obtained by using attributes of each method separately. So, we added to project attributes those which we calculated from the module attributes of each project. The new attributes are the following:

NOC-MENU: total number of menu components
NOC-INPUT: total number of input components
NOC-RQ: total number of report/query components
OPTMENU: total number of menu choices
DATAELEMENT: total number of data elements
RELATION: total number of relations

The statistical study of some of these attributes is shown in figure 1. The projects are between 726 and 8,888 LOC. There are more records with low values of the attributes. However, high values have sufficient weight to influence the induction of the models.

3 Data clustering

All the attributes that are used in this work to generate association rules are numeric, that is, they can take a wide range of values. In order to reduce the number of rules generated it is necessary to divide the range of values of the attributes into a manageable number of intervals**.**

At first, we considered the target attribute LOC (line of code). The values of LOC were partitioned into five uniform intervals. After that, a clustering technique was applied to all available attributes. The

clusters were built by using the single k-means algorithm with a Euclidean distance metric and a weight for LOC-bin (the five LOC intervals) three times greater than for other attributes. This is a way of driving the clustering of the attributes in those LOC intervals. The best results (low overlapping of the LOC intervals) were obtained with five clusters (fig. 2).

From the study of the distribution of attribute values in each cluster we obtained the best partition of the data values in order to generate the association rules. We also observed that the two greater LOC intervals always appeared together in a single cluster. Those intervals represent a low number of transactions, therefore we decided to join them.

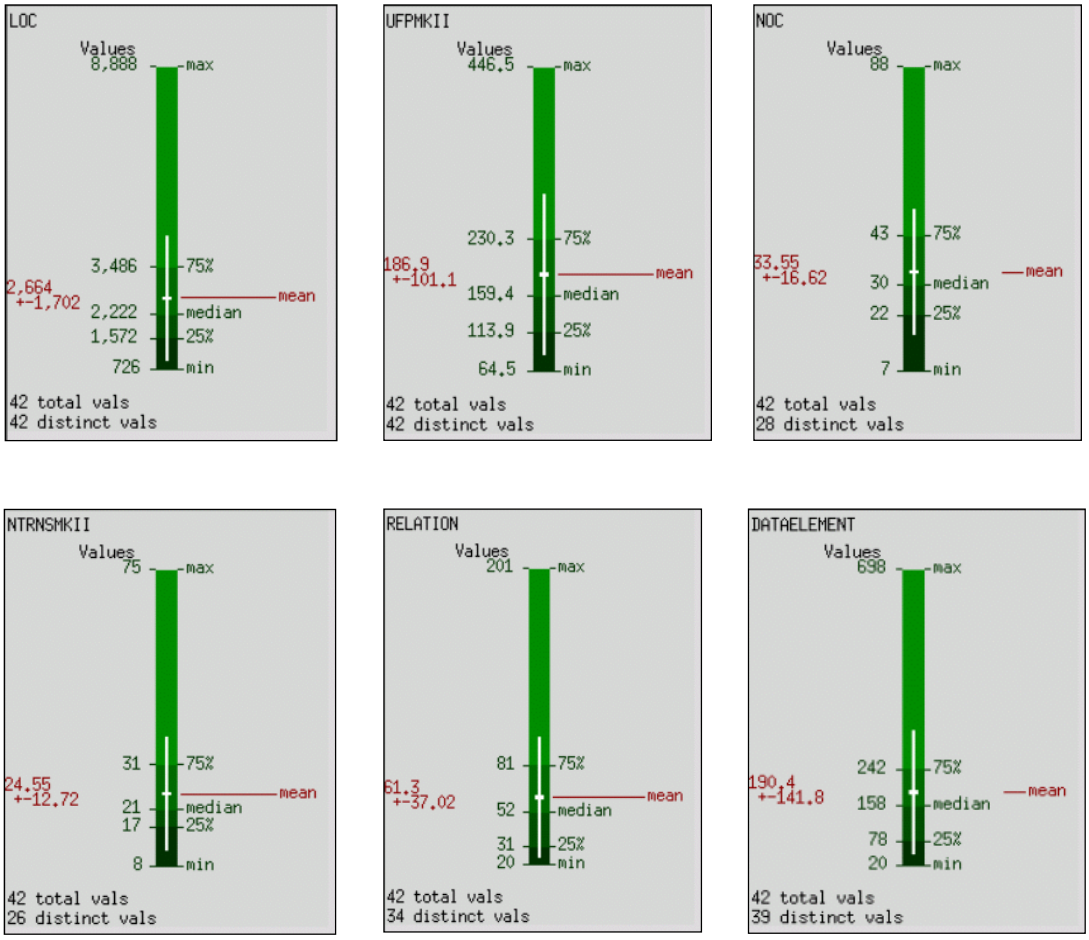

Fig. 1. Statistical study of the data

4 Importance of columns

In classification problems, there is a special attribute called the *label* that one intends to predict. By encoding the relation between the label and the other attributes, the model can make predictions about new, unlabeled data. Importance of columns is a technique that determines how important various attributes (columns) are in discriminating the different values of the label attribute. A measure called *purity* (a number from 0 to 100) informs about how well the columns discriminate the different labels. The cumulative purity measure is a measure of the purity of partitioning the data. The data is partitioned using columns found as important in the same way data is partitioned in a Decision Tree. Each set in the partition has its own purity measure, and the purity measure within the partition is a combination of these individual measures. For a given set in the partition, the purity is 0 if each class has equal representation, and 100 if every record is of the same class. Similarly, the cumulative purity will be 0 if each set in the partition has an equal representation of classes, and 100 if each set in the partition contains record that all have the same class [10].

In our case, we use that method to find the best attributes for discriminating the LOC-bin label. The found attributes were used in the refinement of the association rules.

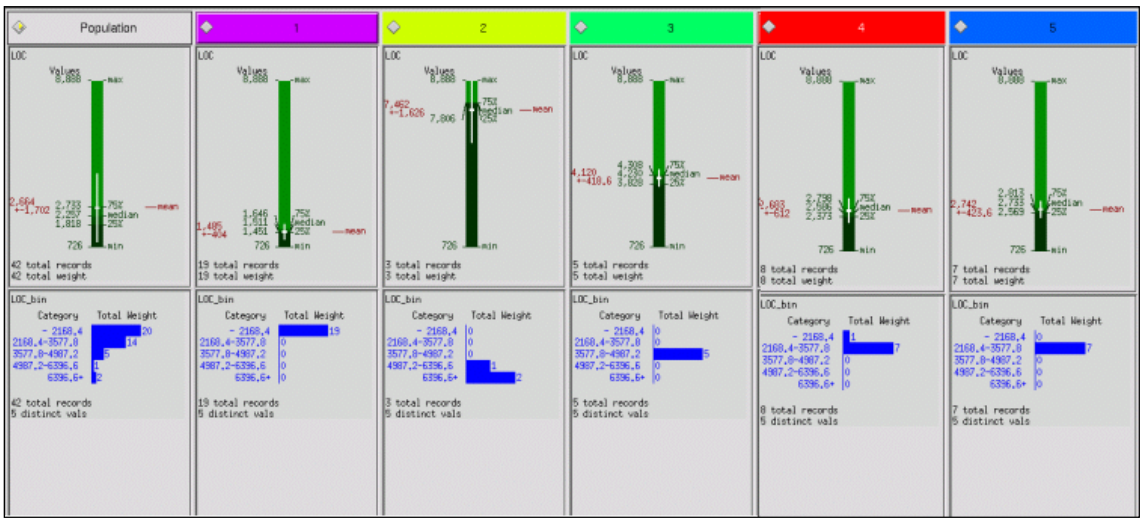

Fig. 2. Distribution of the LOC values in five clusters

5 Decision rules and unexpectedness

Methods for rule discovery are widely used in many domains. However, most of the existing algorithms have the drawbacks that they discover too many patterns which are either obvious or irrelevant. Padmanabhan and Tuzhilin have developed ways to generate useful patterns by incorporating managers' prior knowledge in the process of searching for patterns in data. They generate *unexpected patterns* with respect to managerial intuition and use them to refine domain knowledge [8]. Recently they have proposed an iterative refinement process in which it is possible to search through all possible rules [9]. In this work, we use the concept of *unexpected patterns* to refine decision rules but not their meaning nor their elicitation process. Another difference is the application domain which provides us with another kind of attributes. Padmanabhan and Tuzhilin manage atributes from the business area, most of which are categorical. We use quantitative attributes because they come from the application of software metrics. This fact adds the problem of choosing the best way to partition the range of values of the attributes. In a previous section we explained how to solve it.

Now, we introduce the basis of decision rules and the concepts of unexpectedness given in [9]. Let $I=\{i_1, i_2, \ldots, i_m\}$ be a set of discrete attributes. Let $D=\{T_1, T_2, \ldots, T_N\}$ be a relation consisting on N transactions $T_1, \ldots, T_N$ over the relation schema $\{i_1, i_2, \ldots, i_m\}$. Also, let an atomic condition be a proposition of the form $value_1 \leq attribute \leq value_2$ for ordered attributes and $attribute = value$ for unordered attributes where value, $value_1$ and $value_2$ belong to the set of distinct values taken by attribute in D. Finally, an itemset is a conjunction of atomic conditions. Then we assume that rules and beliefs are defined as extended association rules of the form $X \rightarrow A$, where X is the conjunction of atomic conditions (an itemset) and A is an atomic condition. The strength of the association rule is quantified by the following factors:

Confidence or *predictability*. A rule has confidence c if c% of the transactions in D that contain X also contain A. A rule is said to hold on a dataset D if the confidence of the rule is greater than a user-specified threshold value chosen to be any value greater than 0.5.

Support or *prevalence*. The rule has support s in D if s% of the transactions in D contain both X and A.

Expected predictability. This is the frequency of occurrence of the item A. So the difference between expected predictability and predictability (confidence) is a measure of the change in predictive power due to the presence of X [10].

In [8] unexpectedness is defined by starting with a set of beliefs that represent knowledge about the domain. A rule $A \rightarrow B$ is defined to be unexpected with respect to the belief $X \rightarrow Y$ on the database D if the following conditions hold:

- B and Y logically contradict each other (B AND Y |= FALSE);
- A AND X holds on a ''large'' subset of tuples in D;
- The rule $A, X \rightarrow B$ holds.

Given a belief and a set of unexpected patterns, the motivation behind knowledge refinement is that the belief can be made stronger by refining it based on the discovered unexpected patterns. In the same paper Padmanabhan and Tuzhilin demonstrate formally that the refined rules have more confidence than the original ones.

6 Refinement process

We start with a set of beliefs which relate lines of code with MKII function points. Those beliefs are based on prior knowledge from previous research. Then we search for unexpected patterns that could help us to increase the confidence or to solve ambiguities or inconsistencies between the rules representing the beliefs. The refinement process fits into a generic iterative strategy [9]. Each iteration consists of three steps:

1. Pattern generation procedure: generation of unexpected patterns for a belief.
2. Selection procedure: selection of a subset of unexpected patterns that will be used to refine the belief.
3. Refinement procedure: refining the belief using selected patterns.

The process ends when no more unexpected patterns can be generated. At the end of each iteration, the set of beliefs is checked for validity, where validity may be defined in different terms such as minimum support and confidence.

This generic refinement strategy can be viewed as a broad framework that can allow for a large number of different refinement approaches. We have instantiated this generic strategy and created a specific refinement process that is described below. The principal feature of our approach is the gradual generation of the unexpected patterns by taking a single attribute in each iteration. We take advantage of knowledge of good attributes for classification and use them progressively, beginning with the best. This simplifies the selection of patterns and the refinement process.

Rules representing the beliefs were generated and visualized by using *Mineset*, a Silicon Graphics tool. The initial beliefs are used to seed the search for unexpected patterns. Figure 3 is a graphical representation of first rules on a grid landscape with left-hand side (LHS) items on one axis, and right-hand side (RHS) items on the other. Attributes of a rule (LHS → RHS) are displayed at the junction of its LHS and RHS item. The display include bars, disk and colours whose meaning is given in the graph.

Rules generator does not report rules in which the predictability (confidence) is less than the expected predictability, that is, the result of dividing predictability by expected predictability (pred_div_expect) should be greater than one. Good rules are those with high values of pred_div_expect. We have also specified a minimum predictability threshold of 50%.

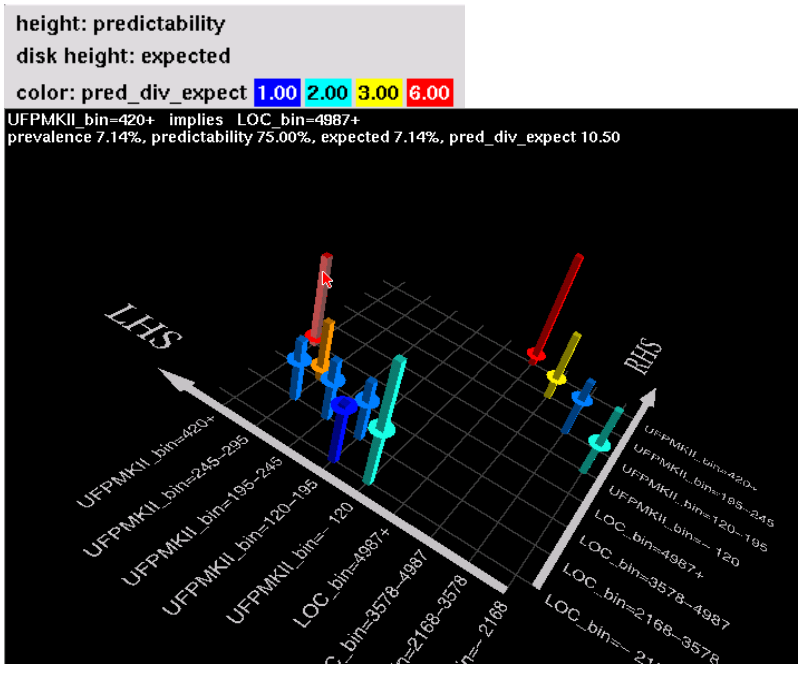

Fig. 3. Rules representing the initial beliefs

The following table contains numeric information about the represented rules. We have extracted the rules with UFPMKII in the left-hand side because UFPMKII serve as predictors of LOC. Relations in the opposite sense are not of interest.

Table 1. Initial beliefs

X (UFPMKII)	Y (LOC)	Confidence (predictability)	Support (prevalence)	Expected predictability	Pred_div_espect
420	>4987	75	7.14	7.14	10.50
245-295	3578-4987	50	4.76	11.90	4.20
245-295	2168-3578	50	4.76	33.33	1.50
190-245	2168-3578	50	9.52	33.33	1.50
120-195	2168-3578	50	16.67	33.33	1.50
120-195	< 2168	50	16.67	47.62	1.05
< 120	< 2168	100	28.57	47.62	2.10

The plot and the table show that good rules are in two software size intervals. When LOC > 4987 the rule confidence is 75%, support 7.14%, and pred_div_espect 10.50. When LOC < 2168 confidence is 100%, support 28.57 and pred_div_expect 10.50. There are less transactions with high values of LOC than with low values, this fact influences the lower value of suppor for LOC > 4987.

Intermediate rules are weak because they have low confidence and show confused correlations between UFPMKII and LOC. Two conflicts that can be observed are the following:

1. UFPMKII: 120-195 → LOC < 2168
   UFPMKII: 120-195 → LOC: 2168-3578
2. UFPMKII: 245-295 → LOC: 2168-3578
   UFPMKII: 245-295 → LOC: 3578-4987

We take these rules (beliefs) to explain the refinement process that we propose. The beliefs selected are those with confidence = 50%. The refinement process is applied to each belief.
Taking as the first belief (X → Y):

If UFPMKII: 120-195 then LOC < 2168
(confidence=50%, support=16.67, pred_div_expect=1.05)

To refine that belief we take the following steps.

1. Pattern generation procedure. Generation of unexpected patterns (A → B) is based on information from classification methods. We select the best attribute for classification (RELATION) and use it to search for rules with LHS fulfilling the condition: LOC≥ 2168. The potential unexpected patterns are shown in fig. 4. The following rule is a unexpected pattern with respect to the belief:

   If RELATION: 62-90 then LOC 2168-3578

   Other possible patterns are the following, but they are not unexpected patterns because the rule A, X → B does not hold.

   If RELATION: 90-153 then LOC 3578-4987
   If RELATION > 153 then LOC > 4987

2. Selection procedure. The unexpected patterns that we choose are those that can be used to solve the conflicts between beliefs. They should have a minimum predictability of 50%. In this example there is only one pattern to be selected:

   If RELATION: 62.6-104.3 then LOC 2168-3578

   This unexpected pattern is used to solve the conflict between the following beliefs:

   If UFPMKII: 120-195 then LOC < 2168 (confidence=50%)
   If UFPMKII: 120-195 then LOC: 2168 - 3578 (confidence=50%)

3. Refinement procedure. The belief is refined by searching for rules with the form X, A → B and X, ¬A→ Y

If UFPMKII: 120-195 and RELATION: 62-90 then LOC: 2168 – 3578
(confidence = 100%, support = 9.52%, pred_div_expect =3)

If UFPMKII: 120-195 and RELATION: < 62 then LOC < 2168
(confidence = 70%, support = 16.67 %, pred_div_expect = 1.47)

Note the significant increase in the confidence and in pred_div_expect.

The remaining beliefs are treated by carrying out the same steps. The process is iterative. Refined beliefs are the input to the next iteration. In that iteration new unexpected patterns are generated by taking another good attribute (the following best attribute) from classification models. The refinements obtained in the first iteration can be seen in figure 5.

In the second iteration the attribute selected to find the unexpected patterns is NTRNSMKII. A belief to be refined in that iteration is:

If UFPMKII: 120-195 and RELATION: < 62 then LOC < 2168
(confidence = 70%, support = 16.67 %, pred_div_expect = 1.47)

The only unexpected pattern with respect to NTRNSMKII is:

If NTRNSMKII: 28-38 then LOC:2168-3578

The refined beliefs are:

X, A →B UFP:120-195, RELATION:62-90, NTRNSMKII:28-38 ⇒ LOC:2168-3578
(confidence: 100%, support: 2.38%, pred_div_expect: 3)

X, ¬A →Y UFP: 120-195, RELATION: < 62, NTRNSMKII:18-28 ⇒ LOC < 2168
(confidence: 75%, support: 7.14%, pred_div_expect: 1.57)

UFP: 120-195, RELATION: < 62, NTRNSMKII<18 ⇒ LOC < 2168
(confidence: 80%, support: 9.5%, pred_div_expect: 1.68)

In all cases confidence and pred_div_expect of the refined beliefs are greater than the original beliefs. Therefore, we have demonstrated empirically that the rules are stronger after the process of incremental refinement.

The iterative process can continue if more unexpected patterns can be generated by taking de following best attribute.

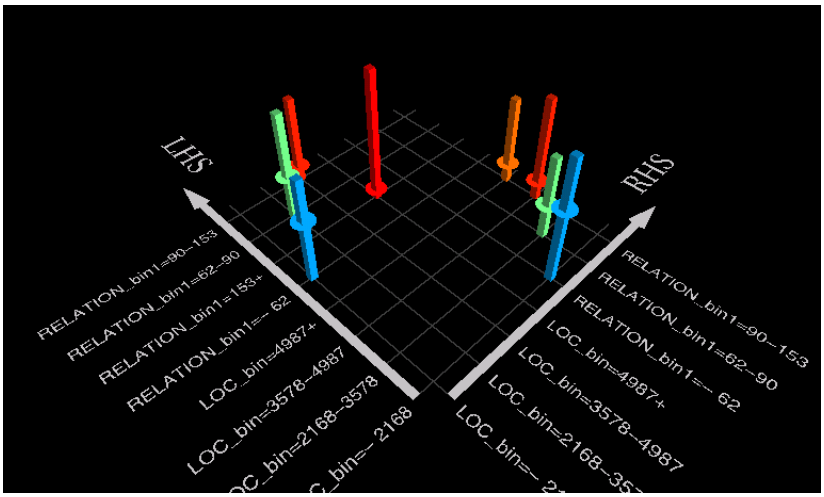

Fig. 4. potential unexpected patterns in the firs iteration

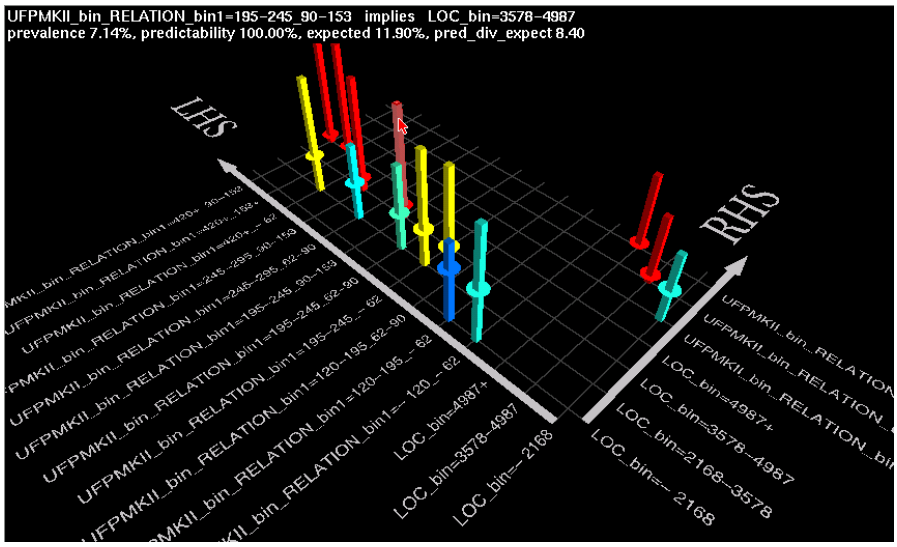

Fig. 5. Refined beliefs in the first iteration

7 Conclusions

Due to the interest that the estimation of the software size has in the first stages of the project, we have studied the relation between early software size measures as the function points and measures of the final product as the lines of code. In this work we have presented the results obtained by using knowledge discovery-driven techniques. The goal is to generate strong association rules between attributes that can be obtained early in the project and the final software size. To do that we have instantiate a rule refinement framework [9] based on discovering unexpected patterns for a belief. By this means we have created a specific process that is very appropriate for solving problems in the project management area. Our approach introduce some features that improve and simplify the refinement process and solve some problems such as the treatment of quantitative attributes.

The main aspects of our proposal are the following ones:

Data clustering is used for partitioning the values of quantitative attributes into a suitable number of intervals. The best partition was obtained from the study of the distribution of attribute values in each cluster

The identification of weak rules representing beliefs and conflicts between them is the starting point to the iterative refinement process.

The generation of the unexpected patterns is gradual, by taking a single attribute in each iteration. We use information about good attributes for classification and take them progressively, beginning with the best. This simplifies the selection of patterns and the refinement process because the number of patterns generated in each iteration is less.

References

1. Albrecht, A.J.: Measuring Application Development. Proc. IBM Applications Development Joint SHARE/GUIDE Symposium, Monterey, CA, (1979) 83-92
2. Hall, B., Orr, G., Reeves, T.E.: A Technique for Function Block Counting. The Journal of System and Software, **57** (2001), 217-220
3. Boehm, B.W., Clark, B., Horowitz E. et al.: Cost Models for Future Life Cycle Processes: COCOMO 2.0. Annals Software Engineering **1** (1995) 1-24
4. Moreno, M.N., Miguel, L.A., García, F.J., Polo, M.J.: Data Mining Approaches for Early Software Size Estimation. Proc. 3rd ACIS International Conference On Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'02), Madrid, Spain (2002). In press
5. Verner, J., Tate, G.: A software size model. IEEE Transaction of Software Engineering, 18 (1992): 265-278
6. Symons, C.R. : Software Sizing and Estimating MKII FPA. John Wiley and Sons, (1991)
7. Dolado, J.J. : A validation of the component-based method for software size estimation. IEEE Transactions on Software Engineering 26(2000): 1006-1021,.

8. Padmanabhan, B., Tuzhilin, A.: Knowledge Refinement based on the discovery of unexpected patterns in datamining. Decision Support Systems 27 (1999) 303– 318.
9. Padmanabhan, B., Tuzhilin, A.: Unexpectedness as a measure of interestingness in knowledge discovery. Decision Support Systems 33 (2002) 309– 321.
10. Mineset user's guide, v. 007-3214-004, 5/98. Silicon Graphics (1998).