Decision Support oriented Enterprise Architecture Metamodel Management using Classification Trees

Ulrik Franke, Johan Ullberg, Teodor Sommestad, Robert Lagerström and Pontus Johnson
Industrial Information and Control Systems
Royal Institute of Technology
100 44 Stockholm, Sweden
{*ulrikf, johanu, teodors, robertl, pj101*}*@ics.kth.se*

Abstract

Models are an integral part of the discipline of Enterprise Architecture (EA). To stay relevant to management decision-making needs, the models need to be based upon suitable metamodels. These metamodels, in turn, need to be properly and continuously maintained. While there exists several methods for metamodel development and maintenance, these typically focus on internal metamodel qualities and meta-model engineering processes, rather than on the actual decision-making needs and their impact on the metamodels used. The present paper employs techniques from information theory and learning classification trees to propose a method for metamodel management based upon the value added by entities and attributes to the decision-making process. This allows for the removal of those metamodel parts that give the least "bang for the bucks" in terms of decision support. The method proposed is illustrated using real data from an ongoing research project on systems modifiability.

Index Terms

Classification trees, Enterprise Architecture, metamodel maintenance, cost-benefit analysis.

1. Introduction

During the last decade, Enterprise Architecture (EA) has grown into an established approach for management of information systems in enterprises. EA is model-based, in the sense that diagrammatic descriptions of the systems and their environment constitute the core of the approach. EA models increase the general understanding of an organization's business and information system landscape and aids in decision making. To ensure semantic rigor, interoperability, and traceability, EA models are typically based upon metamodels. While some of the popular EA frameworks propose a metamodel of their own (e.g. Archimate [1] and the defense frameworks DoDAF [2] and MODAF [3]), others focus on development and maintenance processes that are applicable to many metamodels (e.g. FEA [4] and TOGAF [5]). However, whether explicit or implicit in architecture frameworks, metamodels play an important role in all EA efforts.

Nevertheless, good metamodels are difficult to create and maintain. On the one hand, a metamodel must be suitable to the business and IT decision-making processes it is intended to support. On the other hand, a metamodel must be kept minimal, lest its usage and maintenance will consume too much time and resources within an Enterprise Architecture effort. Thus, there is an inherent trade-off between accuracy and cost, both in the abstract metamodeling phase [6] and in the more concrete data collection phase [7]. In essence, Occam's razor – that entities must not be multiplied beyond necessity – is as valid in metamodeling as in other contexts.

This paper proposes the use of methods from the field of classification tree learning [8] to improve Enterprise Architecture metamodeling in general, and maintenance and re-use in particular. At the core of this approach is a conception of EA as a suitable tool for decision making. Only by creating metamodels designed for specific management decision support can subsequent data collection be kept relevant and cost-efficient. The overall idea, using classification trees to reduce the size of the metamodel (and thereby their maintenance cost) while preserving its predictive power, is illustrated in Figure 1.

The remainder of this paper is structured as follows. Section 2 contrasts the present contribution with some related work, followed by an outline of classification tree learning theory in section 3. Section 4 is the locus of the main contribution, where a number of applications of classification trees to Enterprise Architecture is described. This section also provides detailed examples, based to a large extent on real data, to clearly illustrate the methods proposed. A discussion of the strengths and weaknesses of the proposed method then ensues in section 5, followed by some concluding remarks in section 6.

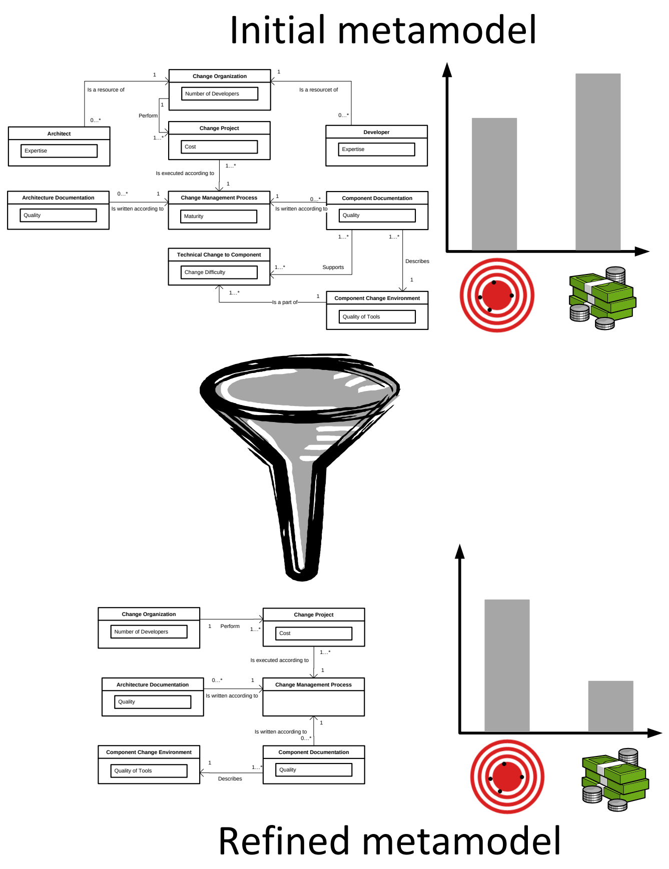


Figure 1. A conceptual sketch of the proposed method, showing how the predictive decision support of a metamodel can be preserved, while the costs are reduced by focusing on the parts of the metamodel with the highest predictive power.

2. Related work

EA relies on the creation of models to represent the enterprise. The concepts represented in this modeling effort are determined by the metamodel used. EA metamodels can serve several different purposes [9]: (i) to document the EA, (ii) to plan and design the EA, and (iii) to analyze the EA. However, regardless of its purpose a metamodel is a trade-off between the precision it gives and the effort required to create and manage its instantiations.

A large number of EA metamodels have been proposed (e.g. [1] [10] [11]) and several frameworks (e.g. [3], [5], [12]) have been developed that contain metamodels or guidance on how to develop them. Although several of these come with descriptions of how to maintain and create models of an EA, they do not detail how to maintain the corresponding metamodels.

A method for analysis of metamodels using classification trees has been presented in [13]. This method applies classification trees for sensitivity analysis of metamodels to test how different variables drive a probabilistic model into different result categories to get a better understanding of the characteristics of the models. However, no prior work has been found that applies classification trees for revising predictive models with the aim of decreasing their complexity and maintainability.

3. Classification trees

A classification tree is a predictive model. In a tree, the *leaves* represent classifications of a target characteristic, the *nodes* represent observations of a specimens, and each *branch* from a node corresponds to a possible observational outcome. The tree structure maps observations to classifications, and can be algorithmically learned from observations.

Most algorithms for learning classification trees employ a top-down greedy search throughout the space of possible classification trees, such as the ID3 algorithm [14] used in this article. The ID3 algorithm is based on selecting the best attribute to observe at each node in the tree. The best attribute is defined as the attribute that best separates the training data according to the target characteristic, using a statistical property called *information gain*. [8]

3.1. Entropy and information gain

Before defining information gain, the *entropy* concept from information theory needs to be defined. Entropy is a measure of the uncertainty associated with a random variable. Given a collection of examples S, where a fraction p_i of the examples takes on each of $1 \ldots c$ different values of the target characteristic, the entropy of S relative to this c-categories classification is defined as [8]:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

With entropy as a measure of the uncertainty, or impurity, in a collection of training examples, it is now possible to define a measure of how effective an attribute is for classifying the training data. This measure, *information gain*, can be explained as the reduction in entropy based on partitioning the examples according to an attribute. Formally, the information gain $Gain(G, A)$ of an attribute A relative to a collection S is given by:

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $Values(A)$ is the set of possible values of an attribute A, and S_v is the subset of S for which it holds that A has the value v. Note that the first term is the entropy of the original collection and the second term is the expected value of the entropy after A has been used to partition

S. $Gain(S, A)$ is thus the expected reduction in entropy achieved by partitioning according to A. [8]

3.2. The ID3 Algorithm

Based on the gain measure, the following is a slightly modified ID3 algorithm for classification tree creation [8]:

```
$ID3(Examples, Target\_attribute, Attributes)$
$root \leftarrow$ empty root node for the tree
**if** all Examples have the same value $c$ of the $Target\_Attribute$ **then**
    **return** $root$ with label $c$.
**end if**
**if** $Attributes = \emptyset$ **then**
    **return** empty $root$.
**end if**
Set $A$ to the attribute of $Attributes$ that has the maximum $gain$.
Set the decision attribute for $Root \leftarrow A$.
**for all** possible values $v_i$ of $A$ **do**
    Add a tree branch to $Root$ corresponding to the test $A = v_i$
    Set $Examples_{v_i}$ to the subset of Examples for which $A = v_i$ holds.
    **if** $\neg(Examples_{v_i} = \emptyset)$ **then**
        below this branch add $ID3(Examples_{v_i}, Target\_attribute, Attributes - \{A\})$
    **else**
        **return** empty $root$
    **end if**
**end for**
```

As compared to the ID3 presented in [8], this version is modified with respect to the branches for which the dataset does not contain any cases (i.e. $Examples_{v_i} = \emptyset$ or $Attributes = \emptyset$). In the original version, the most frequent classification in the whole dataset was used. Here, this is treated as an undefined case, so that the tree faithfully represents the dataset.

4. Applications to Enterprise Architecture Metamodeling

This section will show some application examples for decision trees within the domain of EA. The algorithm outlined above will be used to create decision trees to illustrate the applications.

4.1. Metamodel maintenance

Enterprise Architecture is a holistic approach to information system management and decision making, where models constitute the core. These models are based on metamodels which in turn need to be maintained.

To keep a metamodel aligned with business needs, the information prescribed by the metamodel must be relevant to management decisions. When building a new metamodel, this alignment must be ensured as best possible *a priori*. Metamodel maintenance, on the other hand, has the advantage of hindsight. Historical data can be evaluated *a posteriori* with regard to its effectiveness as decision support. This is where the methods of the previous section enter.

Figure 2 depicts a metamodel designed to support cost prediction of software modification projects. Table 1 is an example of the kind of data that might be available to an enterprise IT decision maker. The data set is taken from an ongoing research project on modifiability [15], [16]. A typical management concern here is to avoid costly future modification projects. To achieve this, the metamodel used should capture precisely as-is those entities and attributes that are relevant to the cost assessment of the to-be state and, ideally, none else.

Table 1 also gives the information gain for each attribute with regard to the cost, i.e. how useful each attribute is for predicting the cost – both on a crude $\{$high, low$\}$ scale and on a finer quartiles scale.

As is evident from Table 1, some attributes by far outdo others in cost predicition. For example, the change difficulties (for components and architectures) have very slight predictive value with a gain of 0.0074. Indeed, one of the most costly (No. 6) and the single cheapest (No. 7) projects both share the same assessment ("Normal") for this attribute. Thus, when maintaining a metamodel by the removal of superfluous entities and attributes, Table 1 provides excellent candidates for deletion, given the cost assessment objective.

4.2. Process generation and tailoring

Continuing the example, it is natural to imagine that the decision maker responsible for software development projects requires not only a metamodel, but also a process of budget control. Information must be put to use once gathered, and this use should be structured so as to consider not only the information gain of single attribute, but also synergies.

The decision trees introduced in the previous section meet these criteria. Assume that the overall management concern is to assess proposed software modification projects by some criteria, and then deliver a verdict; "Yes, go ahead" (if the cost of the project is low enough) or "No, abort" (if the cost is high). Figure 3 illustrates a decision tree based on the data of Table 1, the structure of which is almost such a decision process description. The generation and maintenance of management decision processes, given their goals, can thus be simplified.

Also important is the degree of fit between information need and decision making requirements. Assume that the verdict to be delivered is not two- but four-fold; in addition

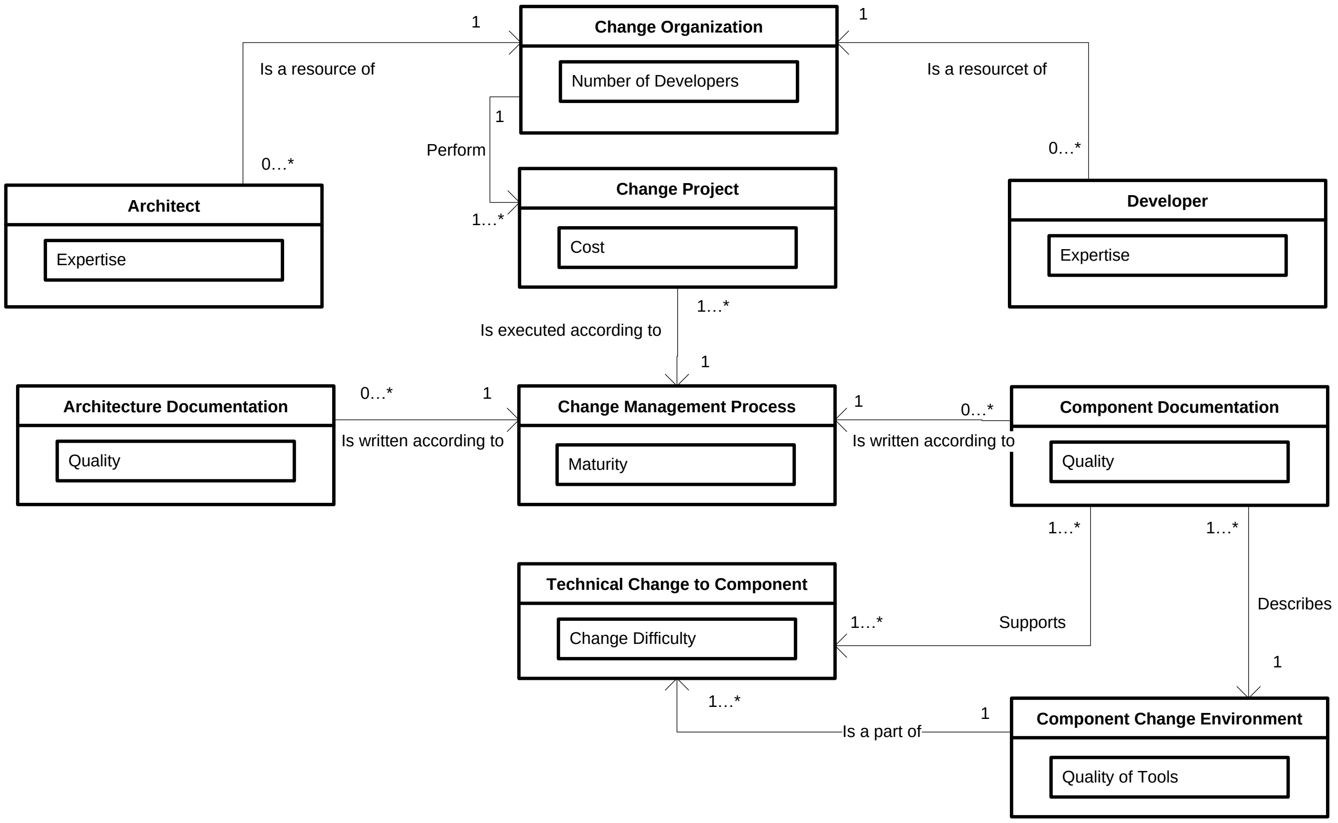


Figure 2. A small metamodel for software modification projects.

Table 1. Costs, attributes, attribute gains for ten software modification projects.

Entities	**Developer Team**	**Component Documentation**	**Component Change Environment**	**Technical Changes to Components**	**Architect Team**	**Architectural Documentation**	**Technical Changes to Architecture**	**Change Organizations**	**Change Management Process**	**Change Project**
Attributes	**Expertise**	**Quality**	**Quality of Tools**	**Change Difficulty**	**Expertise**	**Quality**	**Change Difficulty**	**Number of Developers**	**Maturity**	**Cost**
Project 1	Low	Medium	Low	Difficult	High	Low	Difficult	Many	Mature	20000
Project 2	Low	Medium	Low	Normal	High	Medium	Normal	Medium	Mature	14000
Project 3	Medium	High	High	Normal	Medium	Medium	Normal	Few	Mature	2300
Project 4	High	Medium	Medium	Normal	High	Medium	Normal	Few	Mature	9100
Project 5	Medium	High	High	Normal	Medium	High	Normal	Few	Mature	3000
Project 6	Low	Medium	High	Normal	High	Low	Normal	Few	Not mature	20000
Project 7	High	Medium	Low	Normal	High	Medium	Normal	Few	Mature	1200
Project 8	High	Low	Low	Difficult	High	Low	Difficult	Few	Mature	6228
Project 9	High	High	Medium	Normal	Medium	Medium	Normal	Medium	Mature	2440
Project 10	Medium	Medium	Medium	Normal	Medium	Medium	Normal	Few	Very mature	6266
Gain, medians	0.6464	0.4200	0.0200	0.0074	0.4200	0.1445	0.0074	0.1668	0.2074	
Gain, quartiles	0.8955	0.8200	0.6200	0.1710	0.2955	0.5445	0.1710	0.4058	0.4464	

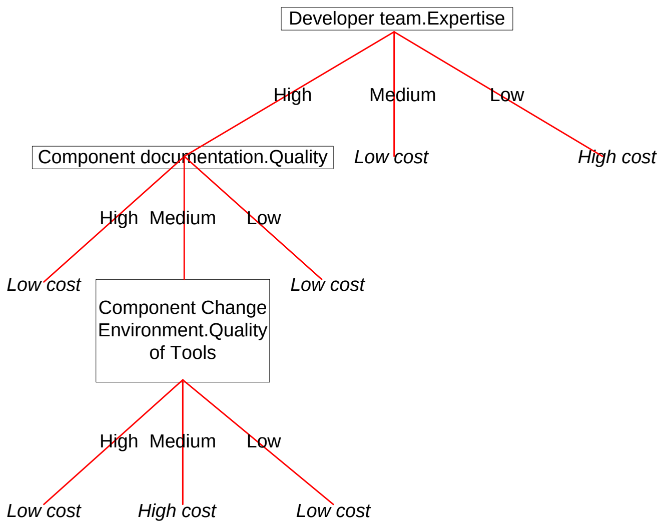
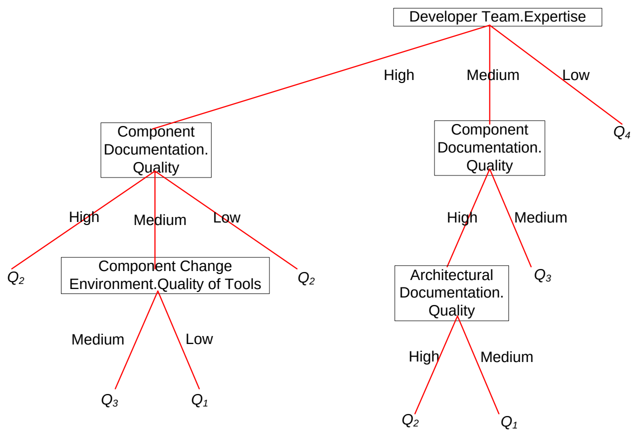


Figure 3. A decision tree for software modification project assessment.



Developer Team.Expertise
High
Medium
Low
Component Documentation. Quality
Component Documentation. Quality
Q_4
High
Medium
Low
Q_2
Component Change Environment.Quality of Tools
Q_2
High
Medium
Architectural Documentation. Quality
Q_3
Medium
Low
Q_3
Q_1
High
Medium
Q_2
Q_1

Figure 4. A decision tree for software modification project assessment, grading costs into quartiles.

to the "Yes" and "No" we add the alternatives "Yes, but with some (cost reducing) changes" and "No, unless major revisions occur". Such additional decision alternatives introduce new requirements on the process and the metamodel. In the example, these requirements can be accommodated by reclassifying the projects into four cost quartiles, $Q_1, \ldots, Q_4$, (where Q_1 denotes the lowest costs and Q_4 the highest), and identify each quartile with a decision, i.e. $Q_1 \rightarrow$ "Yes, go ahead", $Q_2 \rightarrow$ "Yes, but with some changes", etc.

Figure 4 illustrates a decision tree based on these new requirements. It is interesting to note the differences, as compared to the tree of Figure 3. In particular, there is no model monotonicity, e.g. a high level of developer expertise might yield both high and low project costs. Furthermore, there is not necessarily any clear causality in the models: good developers might get the hardest projects and thus the highest costs, while their peers of less expertise get simpler projects and thus lower costs. Clearly, this does not entail that high cost project can be converted into low cost ones by using less experienced developers. Thus, the classification trees capture *correlations* that can be used for prediction, but do not in general yield *causal relations* suitable to act upon.

Three lessons can be learned from this. First, it serves to illustrate the importance of *alignment* between decision making requirements and metamodels. Second, it illustrates the importance of *context* for the relevance of different attributes – information has value not in its own right, but only in a setting. Third, it illustrates the importance of domain knowledge and sanity check when acting on the information.

4.3. Data collection prioritization in large projects

Another application is the optimal use of information when embarking on a large architecture project following a smaller pre-study. In data collection efforts on hundreds or even thousands of IT systems, it is rarely feasible to collect all the data prescribed by a large metamodel. However, if in a smaller pre-study all data is collected for a limited number of systems, the principles outlined above can be used to prioritize before the main data collection phase.

4.4. Cost-benefit trade-offs in modeling

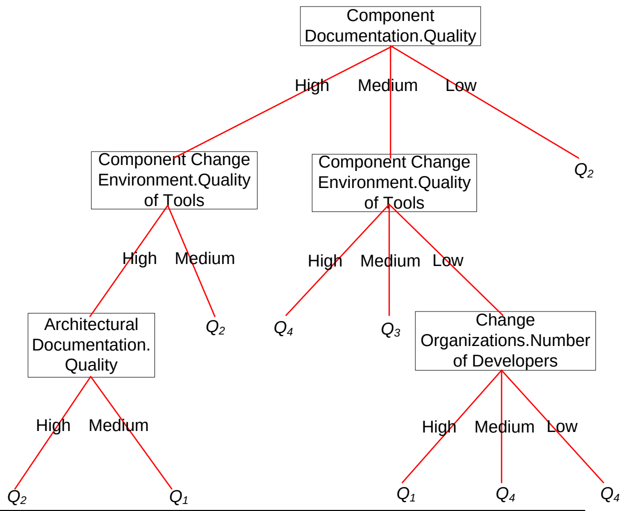
Modeling of large systems may become prohibitively expensive as models grow large. While the actual costs are sometimes unknown, e.g. when embarking on a large data collection project with a fresh metamodel, they are sometimes known. In the metamodel maintenance situation referred to above, financial records or time-tracking of employees might allow the cost of data collection to be discerned. Given such information, it seems prudent to weigh the benefit (information gain) of an attribute against its cost.

There are several ways to incorporate considerations of cost into classification tree learning described in the literature. In the following, we adhere to [17] and replace the standard information gain with the measure

$$\frac{Gain^2(S,A)}{Cost(A)}$$

Figure 5 illustrates the impact on the quartile classification tree of changing the collection costs for the attributes. For the sake of the example, the costs in Figure 5 were obtained as random normally distributed numbers with the cost of the high information gain attribute *expertise of the developer team* multiplied by 10 so it would not be the first choice of the algorithm.

Figure 6 shows a refined metamodel with data collection costs accounted for. Four attributes from the original


Component Documentation.Quality
High
Medium
Low
Component Change Environment.Quality of Tools
Component Change Environment.Quality of Tools
Q_2
High
Medium
High
Medium
Low
Architectural Documentation. Quality
Q_2
Q_4
Q_3
Change Organizations.Number of Developers
High
Medium
High
Medium
Low
Q_2
Q_1
Q_1
Q_4
Q_4

Entity	Attribute	Collection cost
Developer Team	Expertise	5.4657
Component Documentation	Quality	0.4449
Component Change Environment	Quality of Tools	0.6946
Technical Changes to Components	Change Difficulty	0.6213
Architect Team	Expertise	0.7948
Architectural Documentation	Quality	0.9568
Technical Changes to Architecture	Change Difficulty	0.5226
Change Organizations	Number of Developers	0.8801
Change Management Process	Maturity	0.1730

Figure 5. A decision tree using attributes weighted by the attribute data collection costs in the table.

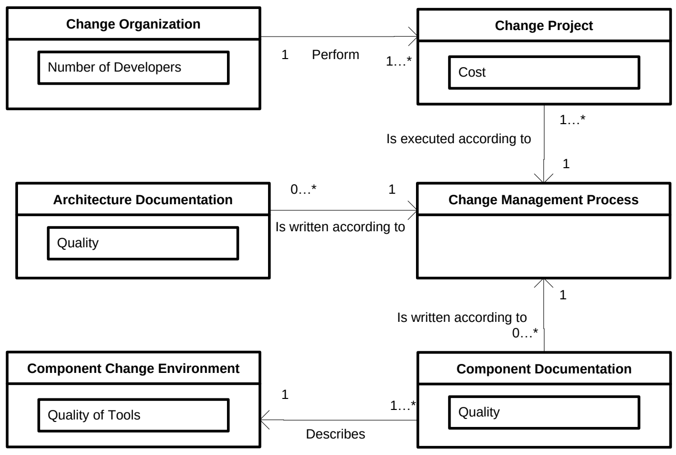


Figure 6. A refined, smaller, version of the metamodel in Figure 2.

metamodel have been removed, as they are not present in the classification tree (depicted in Figure 5). Furthermore, in three cases, the entities holding the attributes, as well as their relationships, could be wholly removed. This example illustrates not only how high costs impact the suitability of a certain metamodel, but also how knowledge of this impact can be used to alleviate the situation. The exact rules for metamodel pruning, given the information gain of its attributes, need to be specified depending on the specific characteristics and purpose of the metamodel.

4.5. Efficient discretization of continuous variables

So far, we have taken for granted that attributes are measured on discrete, not continuous, scales. Such scales can enable lucid overview of attributes and lower data collection cost. The scales considered so far are all *ordinal* scales, i.e. there is an order between the points on the scale (e.g. first low, then medium, then high), but no guarantee that the scales is equally spaced (so that the difference between medium and low is the same as that between high and medium). There are several reasons for the use of ordinal scales. Some attribute scales, such as the maturity of the change management process, are largely determined by their measurement or assessment procedure. Furthermore, measures of process maturity are usually derived from numerous lower-level indicators. Other attribute scales are derived from expert assessments, where scales are unavoidably coarse. When experts assess the difficulty of technical changes to an architecture, it is difficult to imagine scales better than {easy, normal, difficult}. However, there are also cases where a discrete ordinal scale is obviously not the original data format. In our example, the number of developers springs to mind. While it may be appropriate to discretize this number, it is not clear how to best draw the lines between few, medium, and many developers.

The information gain measure offers a solution, tying the discrete scales to the decision at hand. By calculating the information gain associated with different break-points, we can pick the discretization that offers the best decision-making support. The theorem proved in [18], stating that optimal cut points are always to be found on boundaries between two classes (two quartiles in our example), delimits the search. This means that if there is no change of quartile when going from an example e_1 with n_1 developers to another example e_2 with n_2 developers, then there is no use looking for an optimal cut point in the interval $[e_1, e_2]$.

Figure 7 illustrates the search for an optimal discretization of the number of developers. Since we are looking for a {few, medium, many} scale, two cut points are required. These are illustrated on the axes of the graph, thus letting each point in the plane correspond to a discretization. Since the order of the cut points is immaterial, the graph is symmetric and we look only at half of it. Also, since it is a waste to let the two cut points coincide, the diagonal has been excluded from the graph. The best discretization is readily read from the graph, being cut points between 7 and 8, and 14 and 15, respectively.

Table 2 gives the information gain for the old (second column) and new (third column) discretizations. Evidently, the new discretization is a considerable improvement. The intuitive explanation is that the old scale is poorly used,

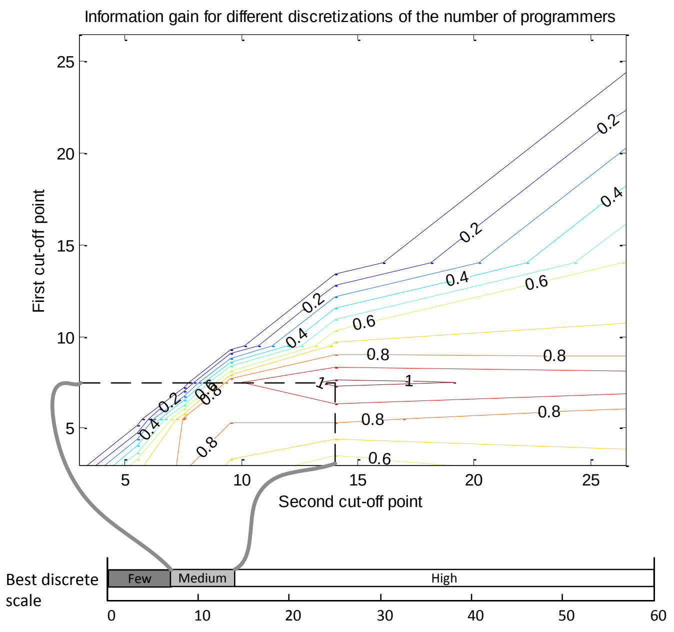


Figure 7. Attribute gains, with respect to quartiles, for different discretizations of the number of developers.

Table 2. Attribute gains for different discretizations.

Number of Developers	$0 \le$ few ≤ 15 $16 \le$ medium ≤ 40 $41 \le$ many	$0 \le$ few ≤ 7 $8 \le$ medium ≤ 14 $15 \le$ many	**Cost of change**
60	Many	Many	Q_4
35	Medium	Many	Q_4
6	Few	Few	Q_1
9	Few	Medium	Q_3
4	Few	Few	Q_2
10	Few	Medium	Q_4
2	Few	Few	Q_1
5	Few	Few	Q_2
18	Medium	Many	Q_2
10	Few	Medium	Q_3
Gain, medians	0.1668	0.4200	
Gain, quartiles	0.4058	1.0200	

having seven specimens classed as low, two as medium and just a single one as high. The new scale is more evenly used. The radical improvement in information gain makes it is reasonable to assume that the number of developers attribute, rescaled, will impact a classification tree using this data. Figure 8 illustrates this, showing that the number of developers attribute now outperforms the developer team expertise attribute, previously the favored root of the tree.

By this method, metamodel attribute scales and data types can be recalibrated to provide optimal decision support.

5. Discussion

There are several issues with the method described in the previous section that require further discussion. The aim of this section is to discuss some such strengths and weaknesses

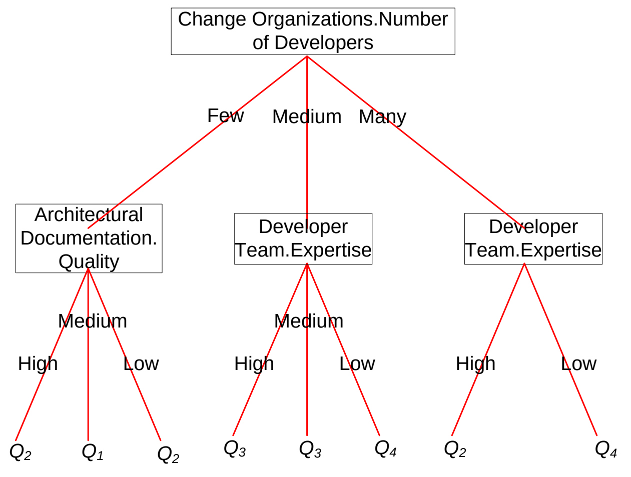


Figure 8. A decision tree for software modification project assessment, grading costs into quartiles using an improved discretization of the number of developers.

of the proposed method.

5.1. Making use of incomplete data

In the examples discussed so far, we have used uniform and complete data sets, i.e. where the attributes are the same and no data is missing in any example. In reality, these conditions are rarely fulfilled. As enterprise metamodels are maintained and updated, old data no longer fits newer. Furthermore, due to lack of time or manpower, immature processes, or simple mistakes, data sets may well contain attributes that have received no values. Nevertheless, IT decisions makers want to make use of such data, since even though it is defective, it still has some predictive value.

Fortunately, the classification tree techniques we have used in the previous section are adjustable to cope with this situation. One strategy [8] is to simply fill use the most common attribute value whenever such a value is missing. More elaborate versions assign the most common value among examples with a common classification [19], or assign probabilities rather than the most common value, as in the C 4.5 algorithm [20].

5.2. Overfitting and the size of data sets

Overfitting is a problem prevalent in machine learning. In essence, the learning process risks going beyond useful generalization and instead learns the peculiarities of the training set. The standard method employed to handle this is to partition the available data into one learning set and another, disjoint, validation set. The tree is then created using the training set, whereafter the validation set is used for checking the validity of the tree. A final tree can then be created by pruning the tree after overfitting [8].

In this article, we make no use of these techniques. The reason is simple – our real world example is too small for them to be useful. With larger data sets, the methods cited above could and should be employed. However, due to the prevalence of small data sets in the world of Enterprise Architecture, a word of caution is appropriate. The presented method has to be applied with caution, as there is no built-in sanity check. Very deep trees based on very small data sets are prone to be erroneous. Sometimes, just looking at information gains might be the best one can do.

6. Conclusions

This paper has shown how techniques from information theory and learning classification trees can be employed to maintain and manage metamodels for Enterprise Architecture-based decision making. The problem is that metamodels tend to grow larger and more difficult to manage as time goes by. The solution is to evaluate metamodel entities and attributes in relation to the decisions that the metamodel is designed to support. Only those parts of the metamodel that provide good "bang for the bucks" decision support should be kept.

In this paper, a method for such metamodel management has been elaborated, and a number of closely related applications have been illustrated and validated by examples based on real data. Furthermore, a thorough discussion of the applicability and limitations of the method has been supplied.

More generally speaking, we have described how a formal analysis method, well established in its own right, can be used to achieve greater analytical capabilities within the Enterprise Architecture discipline. Indeed, there exists a lot of engineering methods, thoroughly explored and validated in different settings, that the cross-disciplinary area of Enterprise Architecture potentially could benefit from using.

References

[1] M. Lankhorst, *Enterprise architecture at work : modelling, communication, and analysis*. Berlin: Springer, 2005.

[2] Department of Defense Architecture Framework Working Group, "DoD Architecture Framework, version 1.5," Department of Defense, USA, Tech. Rep., 2007.

[3] Ministry of Defence, "MOD Architecture Framework version 1.2.003," Available on `http://www.modaf.org.uk`, accessed November 14, 2008, Ministry of Defence, UK, Tech. Rep., Sep. 2008.

[4] Office of Management and Budget, "FEA Consolidated Reference Model Document Version 2.1," Office of Management and Budget, USA, Tech. Rep., 2006.

[5] The Open Group, *TOGAF 2007 edition*. Zaltbommel, Netherlands: Van Haren Publishing, 2008.

[6] U. Franke, P. Johnson, R. Lagerström, J. Ullberg, D. Höök, M. Ekstedt, and J. König, "A method for choosing software assessment measures using bayesian networks and diagnosis," in *Proc. 13th European Conference on Software Maintenance and Reengineering*, Mar. 2009.

[7] P. Närman, P. Johnson, R. Lagerström, U. Franke, and M. Ekstedt, "Data collection prioritization for software quality analysis," in *Electronic Notes in Theoretical Computer Science*, vol. 233, Mar. 2009, pp. 29–42.

[8] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997, ch. 3, pp. 52–80.

[9] S. Kurpjuweit and R. Winter, "Viewpoint-based meta model engineering," in *EMISA 2007: Proceedings of Enterprise Modelling and Information Systems Architecures*. Bonn, Germany: Gesellschaft fr Informatik, Oct. 2007, pp. 143–161.

[10] U. Frank, "Multi-perspective enterprise modeling (memo) - conceptual framework and modeling languages," in *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 3*. Washington, DC, USA: IEEE Computer Society, 2002, p. 72.

[11] K. D. Niemann, *From Enterprise Architecture to IT Governance: Elements of Effective IT Management*. Vieweg: Wiesbaden, 2005.

[12] J. A. Zachman, "A framework for information systems architecture," *IBM Syst. J.*, vol. 26, no. 3, pp. 276–292, 1987.

[13] S. Mishra, N. Deeds, and B. RamaRao, "Application of classification trees in the sensitivity analysis of probabilistic model results," *Reliability Engineering and System Safety*, vol. 79, pp. 123–129(7), February 2003.

[14] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[15] R. Lagerström, "Analyzing system maintainability using enterprise architecture models," *Journal of Enterprise Architecture*, Nov. 2007.

[16] R. Lagerström, P. Johnson, D. Höök, and J. König, "Software Change Project Cost Estimation – A Bayesian Network and a Method for Expert Elicitation," in *Third International Workshop on Software Quality and Maintainability (SQM 2009)*, Mar. 2009, to appear.

[17] M. Tan, "Cost-sensitive learning of classification knowledge and its applications in robotics," *Mach. Learn.*, vol. 13, no. 1, pp. 7–33, 1993.

[18] U. M. Fayyad and K. B. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Mach. Learn.*, vol. 8, no. 1, pp. 87–102, 1992.

[19] J. Mingers, "An empirical comparison of pruning methods for decision tree induction," *Mach. Learn.*, vol. 4, no. 2, pp. 227–243, 1989.

[20] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.