On the Capability of Analysis Techniques in Requirements Engineering

Oscar Dieste Tubío[1] *Marta López Fernández*[2] *Ana Mª Moreno Sánchez-Capuchino*[1]

[1]*Software Engineering Department*
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo s/n, 28660 Boadilla del Monte, Madrid (Spain)
Phone: +34 91 336 {6449, 6929} Fax: +34 91 336.96.17
{odieste, ammoreno}@fi.upm.es

[2]*Computer Science Department*
Facultad de Informática
Universidad de A Coruña
Campus de Elviña s/n, 15071 A Coruña (Spain)
Phone: +34 98 116.71.60 Fax: +34 98 116.71.50 (1238)
martal@udc.es

ABSTRACT

Requirement Engineering methods recommend that particular techniques be used to analyse specific user needs. The capability of these techniques for easing understanding of and representing the user need is questionable, as most of these techniques *are oriented to specific development approaches, that is, they are characterized by their orientation to specific ways of developing software*.

In order to study the development orientation of the different analysis techniques, a framework has been defined that groups the techniques used in the different analysis methods. These techniques have been evaluated according to five criteria: Amplitude, Computational bonds, Procedure of use, Design selection and Design derivation. The result of the evaluation shows that many of the analysis techniques are oriented to a specific software development approach in some degree. As conclusion, none of the technique types provides an adequate basis for supporting the analysis phase.

1. INTRODUCTION

Nowadays, Requirements Engineering (RE) is considered a critical factor in software development. Possibly, the most important activity within this process is to understand the needs raised by the user and to be met by a system under development [Brooks, 87] [Jackson, 95] [Andriole, 96]. Different terms are used in the literature to refer to this activity: Problem Analysis [Davis, 93] [Jalote, 97], Modelling [Loucopoulos, 95] [Somerville, 97], Specification [Wieringa, 95], etc. In this paper, we will adopt Davis's terminology, referring to the activity of understanding and representing user needs as *Problem Analysis*.

The different RE methods recommend that particular techniques be used to analyse specific user needs. These techniques have also been called Conceptual Models (CM) [Dallianis, 92] [Beringer, 96] [Blum, 96]. So, for example, the Structured Analysis (SA) method proposes the use of DFD [DeMarco, 79] [Palmer, 84] [Yourdon, 89]; object-oriented methods (OO), such as UML, propose the use of Object Diagrams (OD) [Larman, 97] [UML, 97], etc.

The capability of these techniques for easing understanding of and representing the user need is questionable, as most of these techniques *are oriented to specific development approaches, that is, they are characterized by their orientation to specific ways of developing software*. For example, the techniques used in OO are conditioned by the use of concepts like classes, inheritance, polymorphism, etc., to develop software systems [Northrop, 97], whereas the techniques used in SA employ concepts as data and transformation functions [Bansler, 93].

This has been pointed out by several authors. For example, Høydalsvik [Høydalsvik, 93] establishes that the techniques used in OO are *Oriented to the Software System*, as they are directly related to the OO development approach. Høydalsvik stresses the need for the use of analysis techniques that are independent of the development approach in software systems development. He refers to such analysis techniques as *Problem-Oriented Methods*. The same approach has been proposed by Loucopoulos [Locopoulos, 95], who claims that the analysis process should necessarily output two product types: *User-Oriented Models* that describe the behaviour and non-functional characteristics of the software and serve as a basis for software engineers, customers and users to understand each other; followed by *Developer-Oriented Models* that specify the functional and non-functional system features and the different types of applicable constraints.

Beringer [Beringer, 95] and Jackson [Jackson, 98] pursue a similar line, stating that current analysis methods focus on the characteristics and structure of the solution rather than on the problem to be solved. This idea has been supported by [Borgida, 85], who claims:

> *"... there is a need for a new class of specification, one that is more oriented to the world of the user than is permitted by current specification methods".*

Note that models of any class act as filters of the real world they represent: they stress the interesting points and hide unimportant details. This essential feature of modelling is also present in the techniques or CMs used in software development. Each technique stresses particular attributes of the information that it represents. So, during problem analysis, software engineers are led by the relationship of dependence between analysis techniques and development approaches. In that way they represent the user need according to the concepts supported by the chosen analysis technique. Thus, these techniques acts as a filter, allowing particular problem issues to be represented and, particularly, those that will be used to develop the system, that is, the representation is conditioned by the peculiarities of each individual approach (SA, OO, real-time systems, etc.).

Furthermore, this dependence between techniques and development approaches means that the same philosophy has to be used throughout all the development phases and, therefore, determines the manner in which the software system is to be built [Henderson-Sellers, 90] [Jalote, 97]. Accordingly, and bearing in mind that each development approach is better suited for addressing a particular spectrum of problems [Davis, 93], there is little prospect of the software engineer switching to a different approach after having analysed the problem, even if the approach selected originally proves not to be the best suited for the need raised at the end of this activity.

All this means that the user need is often adapted to a particular way of building software rather than the artifacts that are best suited for the need raised being used. This redounds upon the quality of the software products developed, as it gratuitously raises software construction complexity, with the resulting impact on both development and maintenance time and costs.

This paper seeks to prove the dependence of analysis techniques on development approaches, for which purpose it presents an evaluation of the techniques or CMs most commonly used today, studying which points of reality they are capable of representing and what associated computational connotations they have. Procedural points related to the use of the analysis techniques are also studied in order to evaluate the guidelines provided to software engineers for representing the user need.

For this purpose, the paper has been structured as follows: section 2 presents the classification framework of the techniques under evaluation; section 3 describes the criteria to be studied for each technique, from the viewpoint of both their expressiveness and their procedural support for use; section 4 shows the results of technique assessment; finally, section 5 presents the major conclusions of this paper, stressing the close relationship between most of the techniques evaluated and particular development approaches. Thus justified, this hypothesis is being used by the authors as a starting point for proposing a redefinition of the problem analysis process, whereby the user need would be studied using analysis techniques that are independent of the development approach.

2. CLASSIFICATION OF ANALYSIS TECHNIQUES

Each analysis method proposes an ordered, though not mechanic, sequence of activities for performing the requirements phase. For example, DeMarco's SA [DeMarco, 79] is a method with a well-defined, well-know sequence of activities: modeling the current physical, current logical, proposed logical and proposed physical system. In the same way, Palmer's SA [Palmer, 84] is also a method, with its own sequence of activities, as Yourdon's SA [Yourdon, 89], OMT [Rumbaugh, 91], Coad's OO [Coad, 90], SADT [Ross, 77], etc. Each method uses one or several analysis techniques, that support the various activities proposed by the different analysis methods. Techniques are recipes, that is, a sequence of procedural steps that actually generates a solution to the problem addressed in a given activity. There are much fewer analysis techniques than analysis methods, as many methods use the same techniques or variations thereon, as shown in figure 1.

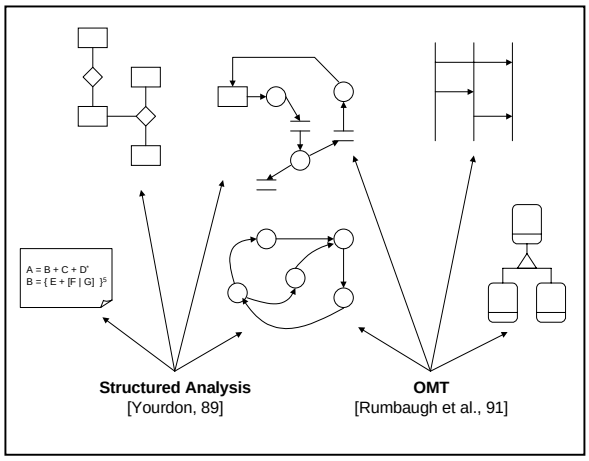

Figure 1. Techniques used in SA and OMT.

Authors like Webster [Webster, 88], Zave [Zave, 90], Bickerton [Bickerton, 93] or Blum [Blum, 96] have proposed classification schemata for some of the existing analysis methods and techniques, based on different criteria. Although they are mostly based on the expressiveness of the different methods, the above criteria do not meet the needs of this paper: show the limitations of the analysis techniques for expressing the user need, independently of any development approach, provide a basis for the discusion about the orientation of analysis methods and techniques and stablish a starting point for the development of new, more adequated, analysis methods and techniques. The main reason is that none of these authors analyse the fitness of the different methods for performing analysis. Additionally, each classification includes different methods and techniques, and it is hard to compare the results of the different authors.

In view of how many methods there are, it is difficult to draw up an alternative classification to those already proposed. This is aggravated by the fact that there is not a set of universally accepted criteria on which the above classification can be

based. Therefore, instead of reclassifying the analysis methods in use today, we opted to analyse and classify the *techniques*, or CMs, used in the different methods. Such analysis and classification is easiest to define and can be used to gain knowledge about the different analysis methods, because methods usually have a **dominant technique**, which drives all later development.

Most analysis methods use different techniques for expressing different viewpoints about the real world [Beringer, 95]. Although a greater wealth of nuances can be expressed by this means, it is very difficult to unify all the information gathered by the different techniques to derive a later design. To bypass this problem, each method has one main technique, and all the remaining techniques are used as a complement. For example, the Object Diagram is the basic technique used in OO; the dominant technique in the case of SA is the DFD and the dominant technique in Information Engineering is the Entity/Relationship Diagram.

The fact that there is a dominant technique does not mean that this is the only technique used. A priori, no method establishes the superiority of one technique over another. This would be to negate, to some extent, the usefulness of the relegated techniques. As mentioned above, each technique provides a particular view of the world; however, the dominant technique acts as a catalyst of all later development, as shown in figure 2, whereas the other techniques support or specify the decisions made on the basis of the dominant technique.

Thus, a method can be defined as **oriented** according to its dominant technique. Thus, for example, a method will be able to be said to be data-oriented if its dominant technique is primarily used to represent the data present in the domain of discourse, or information-transformation-oriented if its dominant technique represents mainly transformation processes.

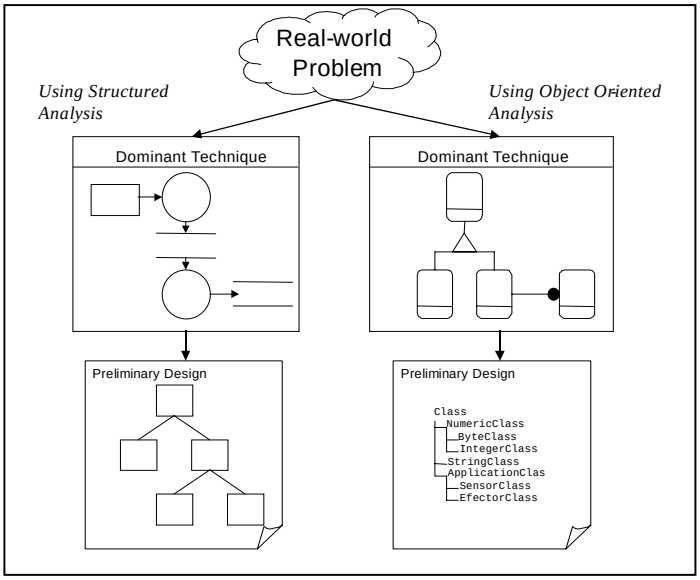

Figure 2. The existence of a dominant technique induces design orientation.

In this work, the authors have analysed and classified analysis techniques, but the results have not been already extrapolated to analysis methods: The classification is as follows:

1. **Procedure-oriented techniques**. Describe the process to be followed to solve a problem or to carry out a task. They replace a natural language text by other less ambiguous representations.

2. **Information-transformation-oriented techniques**. Describe information transformations. Their orientation is similar to the above group, but they are characterized by greater expressiveness, which means that they can express higher level procedures.

3. **Data structure-oriented techniques**. This is the group that is most difficult to define because of the multiplicity of existing techniques. They are characterized by being oriented to identifying the structure of the data in the universe of discourse; however, a host of variants emerge around this central idea. The approach taken in this paper for selecting the techniques included in this group was as follows:

    - Determine which techniques represent the same concepts, even if their diagrams differ, that is, divide the set of existing techniques into classes. The classes were defined depending on the capability of representing the data structure, the capability of defining the operations on the above data and the manner in which these operations are expressed.

    - Determine the best representative of each class identified earlier in view of the relative impact of each technique.

    The above criteria allow a small set of techniques to be selected, which, in essence, faithfully reflect the total set of existing techniques in this category.

4. **Problem structure-oriented techniques**. These could also be termed "Techniques with an explicit problem-domain meta-model". This set of techniques, most of which were developed in this decade, differ from the above in that the underlying ontology is wealthier in terms of concepts. All the techniques in this group explicitly set out the underlying ontology, defining a meta-model, of which each posible model built during analysis is merely an instance.

5. **Dynamics-oriented techniques**. Define the behaviour of a system over time. They are usually used to analyse systems in which control and time are important factors.

6. **Interaction-oriented techniques**. Describe an interchange of information. They consider that user needs and the system can be modelled as a “black box”.

The analysed techniques are presented in figure 3. The techniques used by methods that cannot be classed as being used for analysis purposes, as design or specification methods, have not been included. The fundamental difference between analysis and specification methods has been assumed to be that specification methods are oriented to modelling the knowledge acquired by the software engineer rather than to easing the understanding of the customer/user needs.

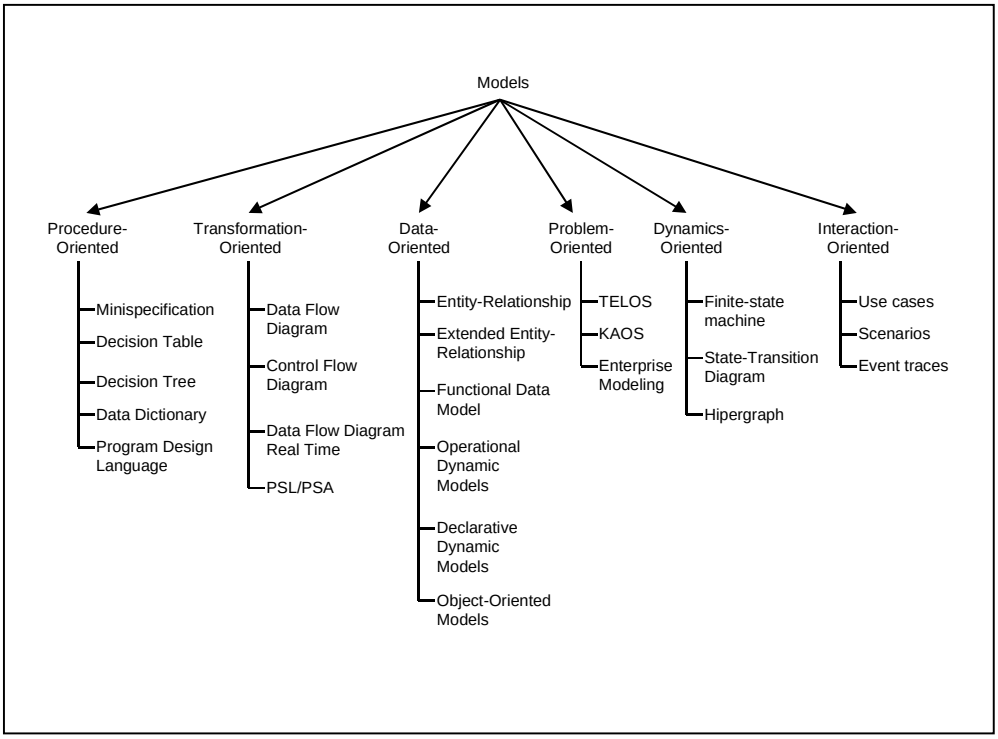

Figure 3. Selected techniques.

3. CRITERIA FOR EVALUATING THE CAPABILITY OF ANALYSIS TECHNIQUES

A series of criteria are needed to be established in order to study the selected techniques. To stablish these criteria, it has been considered that techniques are composed of two elements: a **process** and a **model**. The process is, in principle, a well-defined procedure that guides software engineers in performing the tasks required to solve the problem addressed by the technique. For example, DeMarco [DeMarco, 79] recommends developing a *top-down* DFD, whereas Orr [Orr, 81] establishes a *bottom-up* process. The **model** is a representation formalism, usually graphic, which is used to manipulate the different problem components and document the results. Taking the DFD technique again, the model is a diagram in which the processes are represented as circles, the data as arrows, etc.

The objective of the criteria defined is to allow model expressiveness and the development support provided by the procedural part of the technique to be evaluated. Therefore, two groups have been defined, as shown in figure 4.

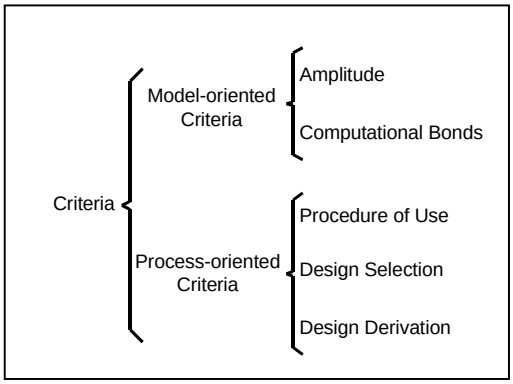

Figure 4. Criteria defined for model evaluation.

Model-oriented criteria refers to model expressiveness. These criteria determine the real-world concepts that each model is capable of representing, and the possibility of entering elements that do not exist in the real world. These elements are mostly computational characteristics, which lead to design considerations being brought into analysis. Two criteria has been defined:

1. **Amplitude**: Determines the capability of the model for representing real-world concepts, that is, how many concepts existing in the universe of discourse can be represented by the model. Notational "tricks" will be required to set out any concepts that cannot be directly represented in the representation formalism. These tricks mostly place constraints on the design of the future system.

   This criterion will have three possible values: **High**, when a lot of real-world concepts can be represented directly, and **Medium**, when the notation has to be constrained or other models have to be used as support for representing a particular concept type. The **Low** value will be used when the model does not explicitly set out real-world concepts, but is used to support another technique (minispecifications, for example, which support the description of DFD processes).

2. **Computational Bonds**: Determines whether the model introduces notations for concepts that do not exist in the problem domain. This criterion determines whether there are elements in the representation formalism that do not match any real-world issue. These concepts are mostly exponents of the underlying computational paradigm and determine a single means of implementing the future system. This criterion has two possible values, which, for the sake of simplicity, will be termed **Yes** and **No**.

The first criterion must not be confused with the second one. Amplitude is used to analyse whether the model allows all the concepts to be represented directly or calls for other models or "tricks" (for example, the use of two processes in a Data Flow Diagram to model a communications channel). The computational bonds criterion analyses whether the model explicitly includes implementation issues (for example, a buffer in the Data Flow Diagram/Real-Time model proposed by [Ward, 85]).

Process-oriented criteria refer to the process by means of which the different models are developed. They also refer to how useful the models are for the subsequent development phases. They, therefore, determine what support is provided to the software engineer for analysis, what possibility there is of their concepts resulting in a design and to what extent they can be transformed into several alternative designs. These criteria are as follows:

1. **Procedure of use**: Determines whether the technique establishes guidelines for performing analysis, that is, the technique has a well-defined process to guide software engineers, especially novices, in performing their work. This criterion will have three possible values: **Doesn't exist**, if the technique does not establish guidelines for performing analysis; **Partial**, if it defines a generic procedure or a series of recommendations, and **Total**, if it gives a detailed process for performing analysis.

2. **Design selection**: Determines whether the technique specifies which design type is the best suited, as Davis [Davis, 93] states, or whether it directly prescribes the use of a particular architecture in the design phase.

   This criterion has four possible values: **No advice**, if the technique does not define which is the best suited design; **Advice**, if it recommends or prioritizes all the possible design methods; **Show**, if it defines which is the most

advisable design, and **Prescribe**, if it univocally determines a design type. The values of this criterion can be ordered from the viewpoint of the support needed by the software engineer as follows: **Show**, **Advice**, **Prescribe** and **No advice**.

3. **Design derivation**: Determines whether the technique establishes a procedure by means of which to derive a design, that is, the technique specifies a series of procedural steps by means of which to derive the design architecture. This criterion has three possible values: **Doesn't exist**, if the technique does not establish guidelines for deriving a design; **Partial**, if it defines a generic procedure or a series of recommendations, and **Total**, if it gives a detailed process for transforming the analysis outputs into design products.

Figure 5 shows the relationship between the established criteria and the major development process phases. The criterion *procedure of use* is related to the process for generating a model of the real world. This model should contain real world concepts (criterion *amplitude*) and can also contain concepts not related to the real world, but to the development world (criterion *computational bonds*). Finally, the criteria *design selection* and *design derivation* refer to the capability of each analysis technique for selecting a computer-based system design and deriving such design.

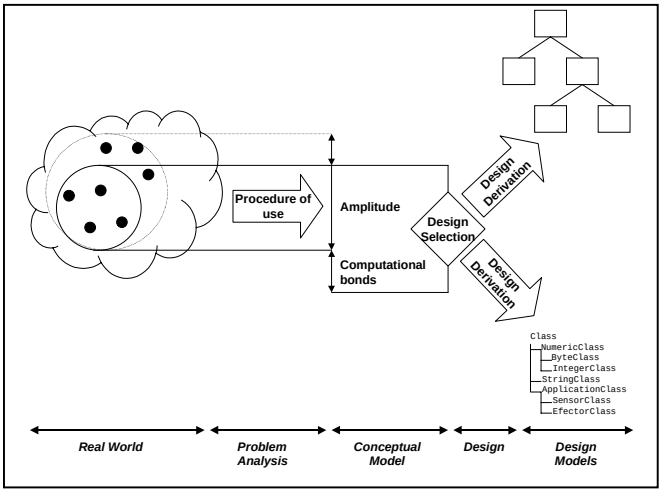

Figure 5. Criteria-development process relationship.

4. EVALUATION OF ANALYSIS TECHNIQUES

Table 1 shows the evaluation of the criteria for the set of selected techniques. As the table provides too much information to gain an overview, the models were regrouped. The objective of this grouping is to determine which techniques are to be considered as solution oriented, although the procedural characteristics to which the second set of criteria used in the evaluation refer also have to be considered to analyse the support that each technique provides to the software engineer during the analysis phase.

GROUP	TECHNIQUE	AMPLITUDE	COMPUTATIONAL BONDS	PROCEDURE OF USE	DESIGN SELECTION	DESIGN DERIVATION
Procedure-	Minispecification	Low	Yes	Doesn't exist	No advice	Doesn't exist

oriented	Decision table	Medium	No	Doesn't exist	No advice	Doesn't exist
	Decision tree	Medium	No	Doesn't exist	No advice	Doesn't exist
	PDL	Low	Yes	Doesn't exist	No advice	Doesn't exist
Transformation -oriented	DFD	Medium	Yes	Partial	Prescribe	Total
	CFD	Medium	Yes	Partial	Prescribe	Doesn't exist
	DFD/RT	Medium	Yes	Partial	Doesn't exist	Doesn't exist
	SADT	High	No	Doesn't exist	Doesn't exist	Doesn't exist
	PSL/PSA	Medium	Yes	Partial	Prescribe	Doesn´t exist
Data-oriented	Data Dictionary	Low	Yes	Doesn´t exist	No advice	Doesn´t exist
	ER	Medium	No	Total	Doesn´t exist	Total
	EER	Medium	No	Total	Doesn´t exist	Total
	FDM	Medium	No	Total	Doesn´t exist	Total
	Operational dynamic models	Medium	Yes	Doesn´t exist	Prescribe	Total
	Declarative dynamic models	High	No	Doesn´t exist	Advice	Partial
	Object-oriented models	Medium	Yes	Doesn´t exist	Prescribe	Total
Domain-oriented	TELOS	High	No	Doesn´t exist	No advice	Doesn´t exist
	KAOS	High	No	Total	No advice	Doesn´t exist
	Enterprise modeling	High	No	Partial	No advice	Doesn´t exist
Dynamics-oriented	Finite-state machine	Medium	No	Doesn´t exist	No advice	Partial
	State-transition diagram	Medium	No	Doesn´t exist	No advice	Partial
	Hipergraphs	Medium	No	Doesn´t exist	No advice	Doesn´t exist
Interaction-oriented	Use cases	Medium	No	Doesn´t exist	No advice	Doesn´t exist
	Scenarios	High	No	Doesn´t exist	No advice	Doesn´t exist
	Event traces	Low	Yes	Doesn´t exist	No advice	Doesn´t exist

Table 1. Result of the Evaluation.

The figure 6 shows the values of the criteria used for regrouping the different techniques. There has not been needed to use all the possible values, or combination of values, of all criteria, but just these that are important to meet the needs of this paper: show the limitations of the analysis techniques for expressing the user need. Thus, it has been considered that a technique with clear computational bonds implies solution-orientation. Also, if any technique prescribes the use of a particular design technology (for example Structured Design, or OO Design), that is, if the value of the criterion *design selection* is *prescribe*, it can be said that such technique has a strong computational background, and should be considered solution-oriented in consequence. On the other hand, a technique is classified as isolated if the criterion *design derivation* evaluates *doesn't exist*, that is, if there are not a process for deriving any type of design.

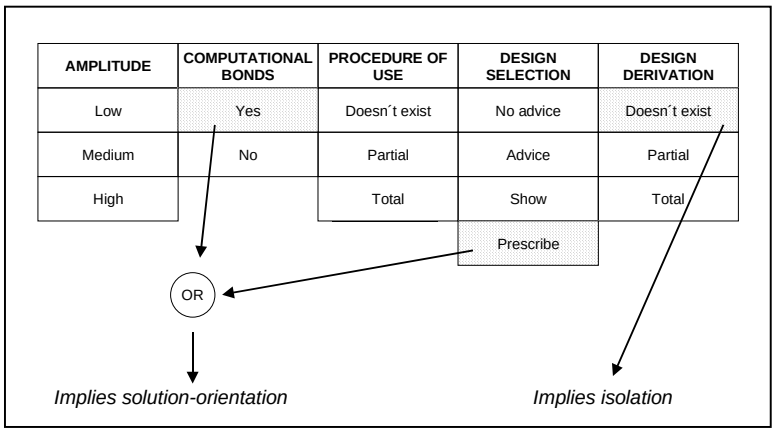

Figure 6. Grouping criteria.

Using the grouping criteria shown in figure 6, most of the techniques fell in two non-disjoint groups. The first group was called "solution-oriented techniques", as the techniques classed therein have computational bonds or directly prescribe the use of a given design type. The second group was called "isolated techniques", as there is no defined procedure for

generating a design of any type from the information contained in the models proposed by the techniques included in this group. The groups defined have the following characteristics:

1. **Solution-oriented techniques**: Oblige analysts to address computational issues during analysis. The models thus developed prescribe a particular design. It will take, at best, a lot of time and effort to transfer the concepts outputted by the analysis phase to an alternative design. This transformation is often out of the question.

   The drawback of using the techniques included in this group is precisely the difficulty in deriving an alternative design. That is, a particular technique for understanding the concepts of the problem to be solved is selected during analysis. If, having completed the analysis and understood the problem, software engineers realize that it is not advisable to design the system as prescribed by the above technique, it is very difficult for them to drop the above approach and, even if it is possible, a switch to another approach will be very costly in terms of effort and time.

2. **Isolated techniques**: Do not determine which sequence of activities have to be performed to derive a design. Their use in the subsequent software development process phases is limited, since, while they provide software engineers with an understanding of the problem, they offer no support for deciding which design to adopt, nor do they allow the above design to be derived.

Most of the techniques analysed fall into the above two groups. This means that these techniques are not suitable for performing the analysis phase, as they either do not allow a design to be generated or they prescribe the use of a single design type, ruling out exploration of alternative development approaches.

Of all the techniques analysed, there is a fraction that cannot be classed in either of the above groups. These techniques have been considered as "problem-oriented" techniques. These techniques have the following characteristics:

3. **Problem-oriented techniques**. The techniques included in this group are really effective for analysing a real-world problem without bringing in clearly computational issues or introducing restraints too early on in the development of a computerized solution. Unfortunately, they can only be used partially in the development process, since either they do not allow the dynamic component of a particular problem to be represented (like ER or FDM models) or they provide a limited capability for deriving a particular design from the information contained in their models.

The allocation of the different techniques to each of the groups identified is shown in figure 7. It follows from figure 6 that the above groups can overlap, that is, they are not disjoint. It is also clear that most of the techniques are either solution-oriented, like the more popular analysis techniques (DFD, OD, etc.), or are isolated and do not provide a procedure for deriving a design.

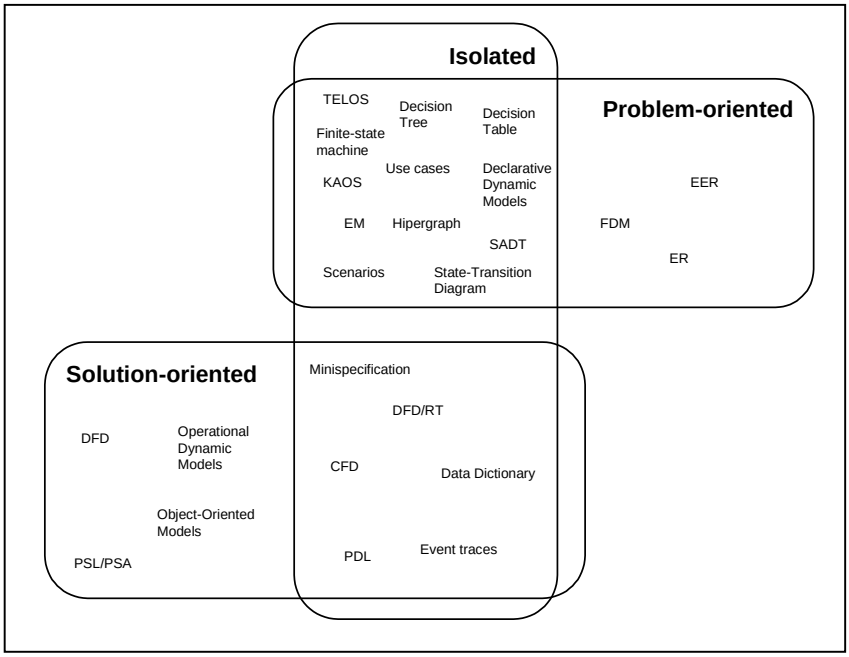

Figure 7. Final classification.

5. CONCLUSIONS

The hyphotesis, already mentioned by other authors, that most analysis methods in use today for understanding the user need are strongly related with the software system that solves those need has been confirmed by the evaluation carried out in this paper. Thus, limitation of the techniques for expressing the user need independently of any development approach has been justified. The most important consequence of this is that the use of actual methods and techniques gratuitously raises software development and maintenance complexity.

For proving the hyphotesis, a classification was defined that groups the techniques used in the different analysis methods. These techniques have been evaluated according to five criteria and, as a result, three technique types were identified. The first type is characterized by being solution oriented. The second technique type is characterized by not providing a mechanism for selecting or deriving a later design. Finally, the third type is characterized by being problem oriented; however, the techniques classed in this group are insufficient either because they do not allow all the real-world concepts to be represented or they do not provide a method for deriving a design.

This evaluation shows that there is a need to define new analysis methods and techniques. These methods must assist software engineers in understanding real-world problems independently of any development approach and evaluating all the alternative system development approaches. The authors of this paper are actually working on this theme.

The results addressed herein are intended to provide a basis for discussion about the orientation of analysis methods and techniques and to serve as starting point in the finding of development approaches closer to the users view of the problem than to the developers view.

6. REFERENCES

[Andriole, 96] Andriole, S.J.; **Managing Systems Requirements: Methods, Tools and Cases**; McGraw-Hill, 1996.

[Bansler, 93] Bansler, J.P., Bødker, K.; **A Reappraisal of Structured Analysis: Design in an Organizational Context;** ACM Transactions of Information Systems, vol. 11, no. 2, 1993, pp. 165-193.

[Beringer, 95] Beringer, D.; **The Model Architecture Frame: Quality Management in a Multi Method Environment**; Proceedings of the SQM'95, Seville, 1995.

[Beringer, 96] Beringer, D.; **The Goals of the Analysis Model**; Technical Report 96/216, Swiss Federal Institute of Technology, 1996.

[Bickerton, 93] Bickerton, M.J., Siddiqi, J.; **The Classification of Requirements Engineering Methods**; IEEE International Symposium in Requirements Engineering, Jan 4-6, San Diego, CA, IEEE Computer Society Press, 1993.

[Blum, 96] Blum, Bruce I. **Beyond Programming. To a New Era of Design**. New York. Oxford university Press, 1996.

[Borgida, 85] Borgida, A., Greenspan, S., Mylopoulos, J. **Knowledge Representation as the Basis for Requirements Specifications**. *IEEE Computer*, vol. 18, no. 4, 1985, pp. 82-91.

[Brooks, 87] Brooks, F.; **No Silver Bullet: Essence and Accidents of Software Engineering**; IEEE Computer, vol. 20, no. 4, 1987, pp. 10-19.

[Coad, 90] Coad, P., Yourdon, E. **Object Oriented Analysis**. Yourdon Press, New York, 1990.

[Dallianis, 92] H. Dallianis. **A Method for Validating a Conceptual Model by Natural Language Discourse Generation**. *Proceedings of the 4th International Conference on Advanced Information Systems Engineering*, Manchester, Great Britain, 1992, pp: 225-244.

[Davis, 93] Davis, A.M.; **Software Requirements: Objects, Functions and States**; Prentice-Hall International, 1993.

[DeMarco, 79] DeMarco, T. **Structured Analysis and System Specification**. New Jersey. Prentice-Hall, 1979.

[Henderson-Sellers, 90] Henderson-Sellers, B., Edwards, J.; **The Object Oriented Systems Life Cycle**; Communications of the ACM, vol. 33, no. 9, 1990, pp. 143-159.

[Høydalsvik, 93] Høydalsvik. G.M., Sindre, G. **On the Purpose of Object Oriented Analysis**. *Proc. of the OOPSLA'93*, pp. 240-255.

[Jackson, 95] Jackson. M.; **Software Requirements & Specifications. A Lexicon of Practice, Principles and Prejudices**; Addison-Wesley, 1995.

[Jackson, 98] Jackson, M.; **Defining a Discipline of Description**; IEEE Software, vol. 15, no. 5, 1998, pp. 14-17.

[Jalote, 97] Jalote, P. **An Integrated Approach to Software Engineering**. New York. Springer-Verlag, 1997.

[Larman, 97] Larman, C; **Applying UML and Patters: And Introduction to Object-Oriented Analysis and Design**; Prentice-Hall, 1997.

[Loucopoulos, 95] Loucopoulos, P., Karakostas, V; **System Requirements Engineering**; McGraw-Hill, 1995.

[Northrop, 97] Northrop, L.M.; **Object-Oriented Development**; in *Software Engineering*. IEEE Computer Society Press, Los Alamitos, USA, 1997, pp. 148-159.

[Orr, 81] Orr, K.; **Structured Requirements Definition**; Ken Orr and Associates, Topeka, Kansas, 1981.

[Palmer, 84] Palmer, J., Mcmenamin, S. **Essential Systems Analysis**. Yourdon Press/Prentice-Hall, New York, 1984.

[Ross, 77] Ross, D.T.; **Structured Analysis: A Language for Communicating Ideas**; IEEE Transactions on Software Engineering, vol. 3, no. 1, 1977, pp 6-15.

[Rumbaugh, 91] Rumbaugh, J. et al. **Object-Oriented Modeling and Design**. Prentice-Hall, New York, 1991.

[Sommerville, 97] Sommerville, I., Sawyer, P.; **Requirements Engineering. A Good Practice Guide**; John Wiley & Sons, 1997.

[UML, 97] UML; **UML Notation Guide**; Version 1.1, September 1997.

[Ward, 85] Ward, P., Mellor, S. **Structured Development for Real-Time Systems**. Vol. 1-3, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[Webster, 88] Webster, D.E; **Mapping the Design Information Representation Terrain**; IEEE Computer, vol. 21, no. 12, 1988, pp. 8-23.

[Wieringa, 95] Wieringa, R.; **Requirements Engineering: Frameworks for Understanding**; John Wiley & Sons, 1995.

[Yourdon, 89] Yourdon, E. **Modern Structured Analysis**. Yourdon Press/Prentice Hall, NY, 1989.

[Zave, 90] Zave, P.; **A Comparison of the Major Approaches to Software Specification and Design**; in *System and Software Requirements Engineering*, Thayer, R.H., Dorfman, M. (Eds), IEEE Computer Society Press, 1990.