Automatic Requirements Elicitation in Agile Processes

Ronit Ankori (Lurya)
Computer Science Dept., Bar Ilan University, Ramat Gan, Israel
luryar@cs.biu.ac.il

Abstract

One of the generic phases of software engineering is the requirements analysis. This paper presents a new method for automatically retrieving functional requirements from the stakeholders using agile processes. The presented system is a machine learning system for the automation of some aspects of the software requirements phase in the software engineering process. This learning system encompasses knowledge acquisition and belief revision in a knowledge base. It is based on Tecuci's multi-strategy task-adaptive learning by justification trees algorithm, known as Disciple-MTL, and supports a few of the practices that Extreme Programming (XP) requires. The aim of the algorithm is to collect information from the various stakeholders and integrate a variety of learning methods in the knowledge acquisition process, while involving certain and plausible reasoning. The result of the manipulation is a list of requirements essential to a software system.

1. Introduction

In the requirements analysis phase, it is essential that we define what it is we plan to develop. This is achieved by interviewing all, or most of, the stakeholders. Stakeholders are people who have an interest in the system, or people who will use the system. They are involved in the definition stage of the system. Different stakeholders can hold different views about what the software should or should not contain. In addition, there are always some vague issues, which were not discussed. The developer cannot and should not guess what to do in these cases. New ways should be thought of, in order to make the analysis as complete as possible.

Automating the process of building knowledge bases has long been the goal of both knowledge acquisition and machine learning. The focus of knowledge acquisition has been to improve and partially automate the acquisition of knowledge from an expert by a knowledge engineer. In contrast, machine learning has focused on mostly autonomous algorithms for acquiring knowledge from data and for knowledge compilation and organization. Automation of knowledge acquisition should be based on a direct interaction between a human expert and a learning system [8], [23].

In the past few years, XP and other Agile Software processes have started to gain considerable impact on software development [6]. The depicted system assists in the integration of XP, by implementing some of its practices. Instead of having one human expert, I am involving all those of interest in the software requirements phase, the stakeholders. The system and the stakeholder work in synergy: the system relies on input from the stakeholder to guide its learning process. Nevertheless, the stakeholder does not need to know much about the knowledge base or enter all the new necessary knowledge explicitly, but can rely on the tool to guide the knowledge acquisition process.

Our system attempts to collect information from the various stakeholders, and manipulate this data. The result of the manipulation is a list of functional requirements essential to a software system. A plausible justification tree [21] is constructed from the information collected from the first stakeholder. Other stakeholders improve, correct and complete the tree, by adding additional pieces of knowledge that extend and add branches to the tree, or remove inconsistent branches. This process continues until there are no more stakeholders and the result tree meets the required standards. The output general rules are then structured from the tree.

Our system supports XP due to the numerous iterations performed and the close collaboration with the stakeholders. It is not easy to decide which of the growing multitude of Agile Processes to deploy in the first place. It requires much experience and a thorough understanding to select and tailor a new process to

Proceedings of the IEEE International Conference on Software - Science, Technology & Engineering (SwSTE'05)
0-7695-2335-8/05 $ 20.00 **IEEE**

IEEE COMPUTER SOCIETY

meet the specific requirements and constraints of a given software project [1], [15]. I hope the system prototyped eases the process.

2. Background

Software engineering is comprised of a set of steps that encompass methods, tools, and procedures, which are referred to as software engineering paradigms. Regardless of the software engineering paradigm, the software development process contains three generic phases. These are definition, development and maintenance. The "definition" phase includes the requirements analysis and specification. Requirements analysis is a software engineering task that bridges the gap between system level software allocation and software design.

As requirements engineering is concerned with the interpretation and translation of the initial informal needs, the elicitation of requirements is perhaps the activity most often regarded as the "first" step in the requirements engineering process.

The requirements analysis task is a process of discovery, refinement, modeling and specification. Both the developer and the customer take an active role in requirements analysis and specification. The customer attempts to reformulate an imprecise concept of software function and performance into concrete detail. The developer acts as interrogator, consultant and problem solver.

Requirements analysis and specification may appear to be a relatively simple task, but appearances are deceiving. The first set of qualities required of specifications is that they should be clear, unambiguous and understandable. A complete understanding of software requirements is essential to the success of a software development effort. No matter how well designed or well coded, a poorly analyzed and specified program will disappoint the user and bring grief to the developer [5], [14].

Knowledge-based systems offer more flexible ways of using a computer than former, standard approaches of programming could do. However, without a means to build up the knowledge base efficiently, the better performance of a knowledge-based system is outweighed by the efforts of constructing and maintaining the knowledge base. Therefore, knowledge acquisition and machine learning have become key issues in artificial intelligence. This process of building a knowledge base underlies all kinds of formalizing applications, whether they are in software or knowledge engineering [11].

XP is a discipline of software development based on values of simplicity, communication, feedback and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation. It is a very social way of developing software, and it is team-oriented. As opposed to other development approaches, XP lacks an investment in up-front design and documentation, but rather encourages communication and discussion [16]. XP builds up to a dozen practices that XP projects should follow [2]. It places a premium on the customer-development relationship, requiring an on-site customer. Both the customer and the developer have clearly defined roles with distinct responsibilities, interacting on a daily basis [25]. Programmers must learn to interact with the customer to discover what is required in user stories.

Applications of machine learning systems show that the initial domain theory provided by the user and possibly completed by the machine learning system tends to be incomplete and inconsistent. Furthermore, machine-learning systems may add incorrect concept definitions they have learned from incorrect example descriptions or insufficient domain theory. This must be solved by automatically revising inconsistent and incomplete domain theories. Some propositional systems were considered, and a few were even implemented [7].

In building business software requirements changes are the norm, the question is what we do about it. One route is to treat changing requirements as the result of poor requirements engineering. Everything in software development depends on the requirements. If you cannot get stable requirements, you cannot get a predictable plan [4], [6].

3. Multi-Strategy Task-Adaptive Learning

A learning process is activated by the input information, obtained from a teacher or from a learner's environment (external or internal). Such a process involves the learner's prior knowledge (background knowledge), and is motivated by the learner's desire to achieve some goal (to solve a problem, to understand given facts or observations, or to perform a task). The learning goal defines the criteria for determining the relevant parts of prior knowledge, choosing the learning strategy, evaluating acquired knowledge, selecting the most preferred hypothesis among the candidate ones, etc. The goal

also plays a major role in determining the amount of effort the learner should extend in pursuing any specific strategy. Thus, learning can be viewed as a process of transforming input information into the desired knowledge by the use of inference and under the guidance of the learner's goal.

The primary inference type performed in any learning act is determined by a "triad" relationship that involves the input, background knowledge and the current learning task. This primary inference defines the "learning strategy". The strategy is inductive, if the input consists of one or more facts, and/or previously generated descriptions existing in the learner's background knowledge, and the task is to generalize the facts and/or improve the descriptions so that the resulting knowledge is useful for solving some new problems. It is analogical, if the input is "similar" to what the learner already knows, and the task is to make a decision about the input that would take advantage of the similar past experience. It is deductive, if the input is principally entailed by what the learner already knows (background knowledge), but the relevant parts of background knowledge are not efficient or directly useful for the learner, and the learning task is to transform these parts into a better form.

In order to learn, an agent needs to be able to perform transformations of knowledge. Research in machine learning has elaborated several single-strategy learning methods like, for instance, empirical induction, explanation-based learning, learning by abduction, learning by analogy, case-based learning, which are based on a primary type of inference and illustrate different ways in which a system can learn [3].

Both empirical learning, which primarily exploits data, and analytical learning, which primarily exploits prior knowledge, are useful methodologies for some domains of application. However, most practical problems seem to fit neither the empirical nor the analytic paradigm. This is because most practical problems involve to a significant extent both prior knowledge and new facts, and the prior knowledge is often incomplete and/or inconsistent.

While past machine learning has been primarily oriented toward single-strategy systems, more recent research has been increasingly concerned with building systems that integrate two or more learning strategies. Single strategies are inherently limited as each of them applies only to a range of problems. Such systems are limited to solving only certain classes of learning problems, defined by the type of input information they can learn from, the type of operations they are able to perform on the given knowledge representation, and the type of output knowledge they can produce. Hence, by properly integrating the various single-strategy methods, one could obtain a synergistic effect in which different strategies mutually support each other and compensate for each other's weaknesses. To extend the capabilities of machine learning programs it is vital to build systems that integrate various strategies. Each of the multi-strategy learning systems that have been built in the last several years illustrates a specific way in which several single-strategy methods could be integrated in order to perform a learning task that could not be performed by a single-strategy method [17].

Due to a complementary nature of many learning strategies, multistrategy systems have a potentially greater ability to solve diverse learning problems than a single-strategy system. On the other hand, because multistrategy systems are more complex, their implementation presents a significant research challenge. An open and ever challenging problem for machine learning research is to develop a system that would integrate a whole spectrum of learning strategies, and would be able to decide by itself which strategy is most suitable in any given learning situation. Humans are clearly able to apply a great variety of learning strategies depending on the problem at hand, and machine-learning system should try to ultimately match this ability.

In multistrategy task-adaptive learning, learning strategies are combined dynamically, according to the task at hand. Research on multistrategy task-adaptive learning attempted to develop a methodology, which for any learning task can recognize what learning strategy, or combination thereof, is likely to be the most effective for solving it (hence, the term "task-adaptive"). The key idea is that any learning process can be viewed as a derivation of desired knowledge from the input information according to the principle of computational economy. What is "desired" knowledge depends on the task the learner wants to perform. How the learner proceeds to obtain this knowledge (learning strategy) depends on what is the most effective way to utilize the available information and the learner's prior knowledge.

The problem of multistrategy learning is not building the mechanisms but their integration and control. The control aspect requires knowledge about the learning strategies: their performance, functionality, representation, accuracy, etc. For multistrategy learning to successfully work, a good understanding of machine learning techniques must be developed. The automatic use of multiple learning strategies and programs for solving hard learning problems is an important problem that attracts new

research effort. Reich in [17] describes a system that uses the manual systematic approach for designing systems that acquire knowledge in complex domains known as M^2LTD. It guides the designer of a large learning program in decomposition, selection and specification procedures. It brings BRIDGER, a system that assists in the design of cable-stayed bridges, as an example for the use of M^2LTD. Two multistrategy systems are incorporated in BRIDGER. The user of one of the systems determines which learning strategies are used in a specific learning scenario, since the automatic control of these strategies is complex.

Michalski in [9] and [10] concentrates on developing an adequate and general computation model for applying a great variety of learning strategies in a flexible and multi goal-oriented fashion and for dynamically accommodating the demands of changing learning situations. He investigates the principles and tradeoffs characterizing diverse learning strategies, attempts to understand their capabilities and interrelationships, to determine conditions for their most effective applicability and to develop a general theory of multistrategy learning.

Our system is based on Disciple. Disciple [20] is an apprenticeship, multi-strategy integrated learning system that can be taught by an expert how to perform domain-specific tasks in a way an apprentice would be taught by the expert. Disciple illustrates a theory and a methodology for learning expert knowledge in the context of an imperfect domain theory. It integrates a learning system and an empty expert system, both using the same knowledge base. It is initially provided with an imperfect (non-homogeneous) domain theory and learns problem solving rules from the problem solving steps received from its expert user, during interactive problem solving sessions. In this way, it evolves from a helpful assistant in problem solving to a genuine expert.

Increasingly complex versions of the Disciple approach have been implemented in the learning apprentice systems Disciple. They have been applied in a variety of domains, such as in the military domain, [18], [24], banking [12] and history education [19], [24].

4. Plausible Justification Trees

A plausible justification tree is a demonstration that the input is a plausible consequence of the knowledge base. The learning method consists of building, generalizing, and/or specializing plausible justification trees of the input examples, and in generalizing and/or specializing the knowledge base so as to entail these trees.

The Multi-strategy Task-adaptive Learning based on building plausible Justification Trees (MTL-JT) integrates deeply and dynamically explanation-based learning, determination-based analogy, empirical induction, constructive induction, and abduction, in order to learn from one or from several positive (and negative) concept examples. The learning task of MTL-JT is both a theory revision task and a concept-learning task.

The framework attempts to automate the multi-strategy process. It proposes that instead of integrating the learning strategies at a macro level, we will integrate the different inference types that generate the individual learning strategies. By this we achieve a deep integration of the learning strategies. The framework also bases learning on building and generalizing a special type explanation structure called plausible justification tree which is composed of different types of inference and relates the learner's knowledge to the input. In this framework, learning consists of extending and/or improving the knowledge base of the system so that to explain the input received from an external source of information.

In a general learning scenario, the system has an incomplete and inconsistent knowledge base (domain model), and it receives new input information about the application domain. The input may take different forms. It may be a ground fact. It may consist of one or several positive and/or negative examples of a concept. It may also consist of one or several positive (and negative) examples of problem solving episodes, each episode specifying a problem and its solution. The goal of the system is to improve its knowledge base so as to consistently integrate the information contained in the input. More precisely, after learning from an input I, the knowledge base should be such that a generalization of I is inferable from the knowledge base. The general learning strategy is based on understanding the input in terms of the current knowledge base. This means that the system will try to build a plausible justification tree that demonstrates that the input is a plausible consequence of the knowledge from the knowledge base. A plausible justification tree is like a proof tree, except that the inferences, which compose it, may be the result of different types of reasoning (not only deductive, but also analogical, abductive, probabilistic, fuzzy, etc.).

An important feature of MTL-JT is that it behaves as a single-strategy learning method whenever the learning task of MTL-JT is specialized to the learning task of the respective single-strategy method. This shows that MTL-JT is a generalization of the

IEEE COMPUTER SOCIETY

integrated learning strategies, which not only takes advantage of their complementarity, but also inherits their features

5. The Methodology

An initial knowledge base is constructed according to the first stakeholder's interview. The information in the knowledge base is incomplete and could perhaps be inconsistent, as there are more stakeholders to be interviewed. Manipulation of this initial data will construct DC – deductive closure. That is, all the data that can be retrieved from the information provided by the first stakeholder using deductive reasoning. This data is then expanded using the various reasoning methods to contain the plausible justification tree built on the initial data. That is the PC – plausible closure. Employing plausible reasoning significantly increases the number of problems that can be solved by the system. However, at the same time, many of the solutions proposed by the system will be wrong. The goal of the algorithm is to extend and correct the knowledge base system until it meets the required specification. Correction of the knowledge base is achieved by adding new facts. The deductive closure of this final knowledge base is called AC – Acceptable deductive closure.

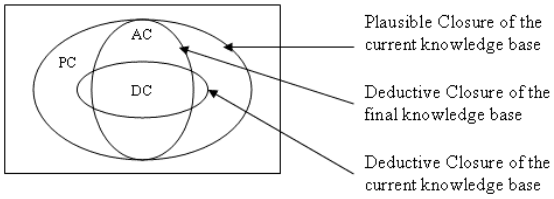

Figure 1. Data manipulation

Multi-strategy task-adaptive learning by justification trees has a great potential for automating the process of building knowledge-based systems. It is a type of learning, which integrates dynamically different learning strategies, depending upon the characteristics of the learning task to be performed. Based on that, a new methodology and tool has been developed called Disciple-MTL, for building verified knowledge based systems, which also have the capability of continuously improving themselves during their normal use. It provides an expert with a powerful methodology and tool for building expert systems, significantly reducing or even eliminating the need of assistance from a knowledge engineer.

The process of building a knowledge base system consists of two stages: Building an initial knowledge base for a general inference engine and developing the knowledge base until it satisfies required specification. During the first stage, the expert generates a list of typical concepts, organizes the concepts, and encodes a set of rules and various correlations between knowledge pieces, which will permit the system to perform various types of plausible reasoning. This initial knowledge base is usually incomplete and partially incorrect. During the second stage, the knowledge base is developed until it becomes complete and correct enough to meet the required specifications.

Employing plausible reasoning significantly increases the number of problems that could be solved by the system. At the same time, however, many of the solutions proposed by the system will be wrong. The goal of the knowledge base development is to extend and correct the knowledge base of the system until it meets the required specifications.

In the first phase, the system performs a complex reasoning process building the most plausible and simple justification tree which shows that the training input indicated by the expert (consisting of a problem P and its expert-recommended solution S) is correct [21], [22]. Because this is the most plausible and simple justification tree, which shows that S is a correct solution for P, the component implications and statements are hypothesized to be true, and are asserted into the knowledge base.

In the second phase, the system reuses the reasoning process performed in the previous phase in order to further extend and also to correct the knowledge base, so as to be able to solve problems from the class of which P is an example, problems similar to the example provided by the expert. The expert will be requested to judge if the results achieved by the system are correct or not. If the answer is correct, the knowledge base will be extended to include the example in it. If the answer is incorrect, the wrong implications made by the system will have to be identified and the knowledge base will be corrected accordingly.

5.1 Building the plausible justification tree

The root of the tree is the input fact (an example of the fact that the expert wants the system to learn). The leaves are facts from the knowledge base, and the intermediate nodes are intermediate facts generated during the understanding process. The branches connected to any given node link this node with facts, the conjunction of which *certainly* or *plausible implies* the fact at the node, according to the learner's knowledge base. The notion "plausibly implies" means that the target (parent node) can be inferred

IEEE COMPUTER SOCIETY

from the premises (children nodes) by some form of plausible reasoning using the learner's knowledge base. The branches together with the nodes they link represent individual inference steps, which could be the result of different types of reasoning.

The method for building such a tree is conducting top-down uniform cost search [13] in the spaces of all AND trees, which have a depth of at most p, where p is a parameter, set by the expert. The cost of a partial AND tree is computed as a tupple (m, n), where m represents the number of the deductive inference steps in the tree, and n represents the number of the non-deductive inference steps. The ordering relationship for the cost function is defined as follows: (m1, n1) < (m2, n2) if and only if n1<n2 or (n1=n2 and m1<m2). This guarantees to find deductive tree, if exists, with the fewest inference steps.

When considering new example, the system tries to generalize the current justification tree so as to cover a justification of the new positive example. At the same time, it may also generalize the knowledge base if this is needed in order to entail the new tree.

First of all, the system determines the instance of the general tree corresponding to the new example. The system then analyzes the leaf predicates and the inference steps from this proof tree. If the leaf predicates are true and the inference steps are plausible, then the tree is a plausible justification of the new positive example that is already covered by the general justification tree (original). This ends the processing of the current example. That is, the new example is already covered by the plausible tree that we have, so no further processing is required. However, if this is not the case, a reasoning method should be applied.

The next step of the learning process is to build the explanation structure, which has three general components to be unified:

- The part of the tree from the generalized justification tree of the knowledge base that covers part of the plausible justification tree of the example.
- The part that is specific to the generalized justification tree of the knowledge base.
- The generalization of the part of the tree in the example that is specific to it.

The initial tree building is time consuming. The generation of the addition is simple matching and instantiation process, and is therefore rather quick. The plausible justification tree is generalized by generalizing each implication and by globally unifying all these generalizations.

An important feature of the presented method is that it behaves as a single-strategy learning method whenever the applicability conditions of such a method are satisfied, and the learning task of MTL is specialized to the learning task of the single strategy method. This feature is important because it shows that the MTL method is a generalization of the integrated learning strategies that not only takes advantage of the complementarity of the integrated strategies, but also inherits the features of theses strategies.

6. Experimentation

As a concrete example, I have chosen to deal with the functional requirements for building a self-corporate care system for the telephony industry. Communication service providers today face an enormous challenge in managing relationships with large corporate customers. Corporate customers are more complex to manage, and require more attention and a high level of service. With advanced Web self-service technologies, corporate customers can manage their own users, equipment, services, bills, orders and trouble tickets. It is thus imperative to have an appropriate self-service framework to support the unique and complex needs of corporate customers.

The self-corporate system allows the service provider to empower large customers such as college campuses, hotels or large industrial companies, to support themselves over the Web. It also enables the corporate customer's telephone departments and other administrators to manage their own internal telecommunication operations. This allows them to take responsibility for level one support, i.e., setting up a new phone, disconnecting, basic changes (new service), replacing bad equipment, etc., while the telecommunication company is responsible for level two support, i.e., actually fixing the equipment, solving service and provisioning problems, etc.

The system enables customers to have a significant amount of management functionality over their telephone resources. By providing the customer with this level of control over internal changes and internal services provisions, the service provider can reduce the level of effort required in servicing the customer, while enhancing customer support and promoting customer satisfaction.

Out of this domain I have selected about a dozen requirements (figure 2).

1. System should be able to manipulate general account information (change or delete personal data or address details).
2. System should be able to manipulate password (set/change/reset).
3. System should allow viewing of account information using web interface.
4. System should allow exporting account information to MS-Office applications.
5. System should allow exporting account information to PDF format.
6. System should allow viewing bill summary statistics using web interface.
7. System should allow exporting bill summary statistics to MS-Office applications.
8. System should be able to generate bill summary statistics in PDF format.
9. System should allow viewing bill statistics using web interface.
10. System should allow exporting bill statistics to MS-Office applications.
11. System should be able to generate bill statistics in PDF format.
12. System should allow modification of trouble tickets.
13. System should allow to remove/delete trouble tickets.

Figure 2. Requirements elicited from stakeholder

Each of the requirements was depicted as a Horn Clause that describes general rules and facts (figure 3).

```
manipulate(X) :- perform_change(X), perform_delete(X).
view(X,Y) :- isa(X,data), isa(Y,application).
supports(X,required-formats) :- view(X,web),
                view(X,pdf), view(X,ms-office).
manage(X) :- allows_to_view(X,Y), manipulate(Y),
                supports(Y,required-formats).
perform_change (trouble-ticket).
perform_change(account-info).
peform_delete(trouble-ticket).
view(bill-statistics,pdf).
view(bill_summary_statistics,pdf).
view(account-info,ms-office).
view(account-info,pdf).
view(bill_summary_statistics,web).
view(bill_statistics,web).
allows_to_view(account,account-info).
```

Figure 3. Initial knowledge base generated

These requirements served to build the initial knowledge base. This data was allegedly retrieved from the first stakeholder. The algorithm was then applied to these requirements and a set of final requirements was generated. Figure 4 describes a plausible justification tree for a new fact manage(account). We can see that the tree generated the new plausible rules:

perform_change(Y):~perform_delete(Y).
view(Y,pdf):~view(Y,web).

7. Conclusions

Properly defining the requirements is a crucial and integral part of the requirements analysis phase. Knowing what it is we want to develop and correctly defining it plays an important role in the software engineering field. This paper aims to automate and ease the requirements elicitation process by describing a machine learning system using a knowledge base that applies some of the practices of Extreme Programming. The system also supports elements of knowledge acquisition and knowledge refinement. Our system demonstrates the use of the MTL-JT algorithm in the field of software engineering, a field in which it was not previously implemented. It was able to test sample requirements from various stakeholders and produce a set of requirements that can serve as final requirements.

8. Acknowledgements

The author of this paper wishes to thank Prof. Uri Schild for his continuous support and belief.

9. References

[1] Bailey, P., Ashworth, N., Wallace, N., "Challenges for Stakeholders in Adopting XP", in *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002), 2002.*

[2] Beck, K. *Extreme Programming Explained*, Addison-Wesley, 2000.

[3] Carbonell, J.G., Michalski, R.S., Mitchell, T.M., "An Overview of Machine Learning", in Carbonell, J.G., Michalski, R.S., Mitchell, T.M. (editors), *Machine Learning: An Artificial Intelligence Approach,* Tioga Publishing Company, Palo Alto, CA, 1983.

[4] Fowler, M., "The New Methodology", online at: http://www.martinfowler.com/articles/newMethodology.html , April, 2003.

[5] Ghezzi, C, Jazayeri, M., Mandrioli D., *Fundamentals of Software Engineering*, Prentice Hall, second edition, New Jersey, 2003.

IEEE COMPUTER SOCIETY

[6] Heinecke, H., Noack, C., Schweizer, D., "Constructing Agile Software Processes", in *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002), 2002.*

[7] Kodratoff, Y., "Industrial Application of ML: Illustrations for the KAML Dilemma and the CBR Dream" in Luc de Raedt and Francesco Bergadano (editors), *Machine Learning: ECML-94, volume 784 of Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 3-19, 1994.

[8] Langley, P., *Elements of Machine Learning*, Morgan Kaufman Publishers, San Francisco, CA, 1996.

[9] Michalski, R.S., "Toward a Unified Theory of Learning: Multistrategy Task-Adaptive Learning", *Reports of Machine Learning and Inference Laboratory, MLI 90-1, Center for Artificial Intelligence*, George Mason University, Fairfax, VA, 1990.

[10] Michalski, R.S., "Inferential Theory of Learning: Developing Foundations for Multistrategy Learning", in Michalski, R.S. and Tecuci, G. (Editors), *Machine Learning: A Multistrategy Approach, Vol. 4,* Morgan Kaufman, San Mateo, CA, 1994.

[11] Morik, K., Wrobel, S., Kietz, J.U., Emde W., *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications,* Academic Press, London, 1993.

[12] Nedellec C., Correia J., Ferreira J. L., Costa E., "Machine Learning goes to the Bank," *Applied Artificial Intelligence*, special issue on applications of ML, 1994.

[13] Nilsson, N., *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, 1971.

[14] Pressman R.S., *Software Engineering: Practitioner's Approach,* McGraw-Hill, third edition, Berkshire, 1994.

[15] Radding, A., "Extremely Agile Programming", *ComputerWorld,* Online at: http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,67950,00.html, February, 2002.

[16] Rasmusson, J., "Introducing XP into Greenfiled Projects: Lessons Learned", *IEEE Software,* IEEE Computer Society, California, February, 2002, pp. 21-28.

[17] Reich, Y., "Macro and Micro Perspectives of Multistrategy Learning", in Michalski, R.S. and Tecuci, G. (Editors), *Machine Learning: A Multistrategy Approach, Vol. 4.* Morgan Kaufman, San Mateo, CA, 1994.

[18] Tecuci, G., Hieb, M.R., "Teaching Intelligent Agents: the Disciple Approach", *International Journal of Human Computer Interaction*, 1996, Vol. 8, No. 3, pp. 259-285.

[19] Tecuci, G., Keeling, H., "Teaching an Agent to Test Students" in *Proc. 15th International Conf. on Machine Learning*, Morgan Kaufman, San Francisco, CA, 1998, pp. 565-573.

[20] Tecuci, G., Kodratoff, Y., "Apprenticeship Learning in Imperfect Domain Theories", in Kodratoff Y. and Michalski, R.S. (Editors), *Machine Learning: An Artificial Intelligence Approach, Vol. 3.* Morgan Kaufman, San Mateo, CA, 1990.

[21] Tecuci, G., "Plausible justification trees: a framework for the deep and dynamic integration of learning strategies", *Machine Learning*, 1993, 11:237-261.

[22] Tecuci, G., "An Inference-Based Framework for Multistrategy Learning", in Michalski, R.S. and Tecuci, G. (Editors), *Machine Learning: A Multistrategy Approach,* Morgan Kaufman, San Mateo, CA, 1994.

[23] Tecuci, G., "Building knowledge bases through multi-strategy learning and knowledge acquisition", in Tecuci, G. and Kodratoff, Y. (Editors), *Machine Learning and Knowledge Acquisition: Integrated Approaches,* Academic Press, London, 1995.

[24] Tecuci, G. (with contributions from the LALAB members Dybala T., Hieb M., Keeling H., Wright K., Loustaunau P., Hille D., Lee S.W.), *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies,* Academic Press, London, 1998.

[25] Wood, W., Kleb, W. L., "Exploring XP for Scientific Research", *IEEE Software,* IEEE Computer Society, CA, May-June 2003, pp. 30-36.

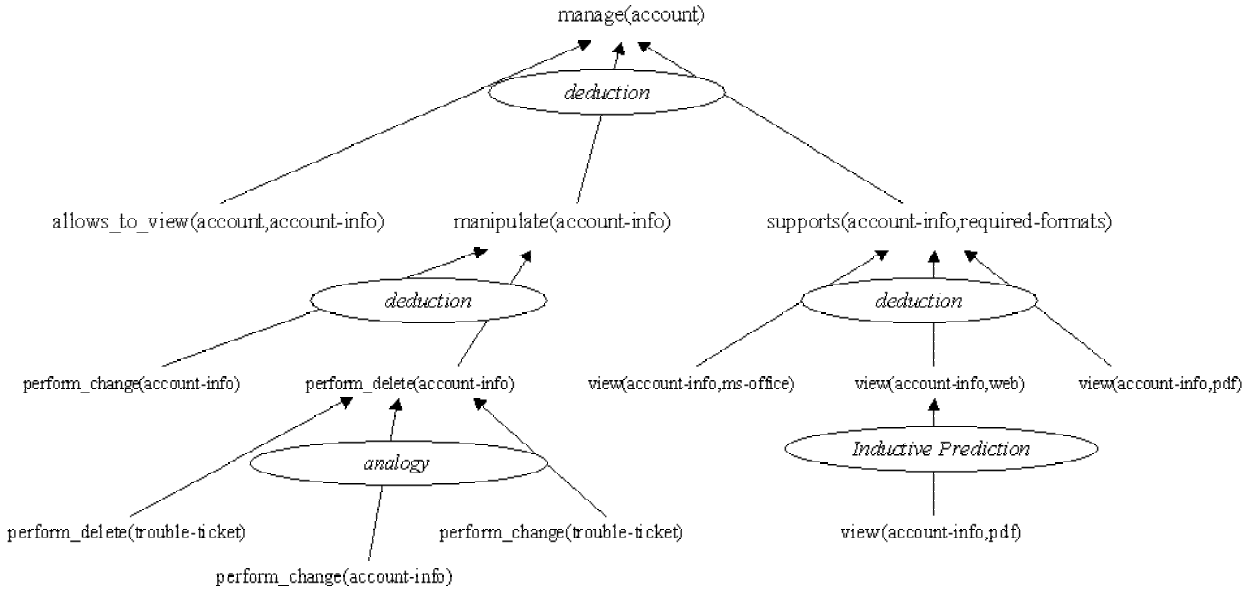

Figure 4. Plausible justification tree for manage(account)

IEEE COMPUTER SOCIETY