A Logic-Based Framework for Reasoning Support in Software Evolution

Vassilios C. Vescoukis, Nikolaos Papaspyrou and Emmanuel Skordalakis

{bxb, nickie, skordala}@softlab.ntua.gr

National Technical University of Athens
Electrical & Computer Engineering Department
Computer Science Division, Software Engineering Laboratory

Abstract. Software development has been acknowledged as a complicated problem-solving activity done in a complex, multi-dimensional space. People actively involved in software development need support in understanding and documenting not only the description of the software developed, but also the problem domain and the reasons behind decisions taken during evolution. Development methods do not provide such support, and researchers begin to explore the recording of reasoning in specific phases of software evolution. Several data models have been presented, aiming to support developers by maintaining a repository of deliberation elements of decisions taken during development; these models are usually supported by a special CASE tool that can be classified as a Software decision management system. Using experience gained in the development of large-scale applications in an industrial environment, and experimenting with prototype software design decision management systems, we introduce PROFILE, a conceptual logic-based meta-model that integrates software evolution process modelling with deliberation representation and decision factors. PROFILE enhances existing models by providing support throughout the whole software evolution, by capturing assumptions as real-world reasoning elements, and by maintaining a Knowledge Base of decision-making factors. A prototype Software Evolution Decision Support System based on an instance of PROFILE is also presented in this paper.

Keywords: meta-models, decision-based software development, logic-based models, knowledge-based software engineering, conceptual modelling, reasoning, software evolution support.

1. Introduction

For the past 30 years there has been an intensive effort to improve the software development process by resolving what has been called *software crisis*. Software development methods have been introduced to help produce "good" software, focusing on *how* this should be done. As they where introduced, new development methods appeared to be very promising, but soon enthusiasm was replaced by realism, and yet today, there is no doubt that *the crisis is still here.*

Documentation is the only means stakeholders have for understanding a software system. It must, therefore, be as complete, understandable and easy-to-communicate, as possible. Practice has shown that *everything* changes in the time allocated to a software development effort: requirements are never the same as they were when the development begun, assumptions and constraints change, too. It is the real world that

changes, which has an effect in software and these changes must be reflected in software documentation.

We can consider *documentation* as the result of a problem-solving activity defined by a software development method. It is an answer to a "*what needs to be done...*" or "*how should something be done...*" question. Experience shows that this kind of documentation is not equally well understood by all stakeholders involved in software development. *Users* familiar to the problem domain, *might* understand some "what-type" answers, provided that they are properly expressed. *Software engineers* are supposed to understand everything, but this claim cannot be supported in a "court of realism". *Managers* do not need, nor do they want to understand such documentation. Finally, *marketeers* usually derive their own version of documentation which is not necessarily exact or up-to-date.

For the past few years, research has focused on reasoning in software development. During software development several artifacts are produced; before the production of each artifact a deliberation takes place (Figure 1). Several conceptual models have been introduced to capture elements of this deliberation either in the software requirements specification phase, or in the software design phase. A new dimension in software documentation has been introduced. Researchers try to enhance documentation by giving answers to "*why-type*" questions, which is believed that will make documentation more useful, conceivable and easy-to-communicate. So far, focus is on two discrete phases in software development: requirements engineering and design. New data models and tools have been developed to support those involved in these phases of development, capture and exploit reasoning in their work. However, little has been published about experience in using such tools and on practical results of the proposed approaches.

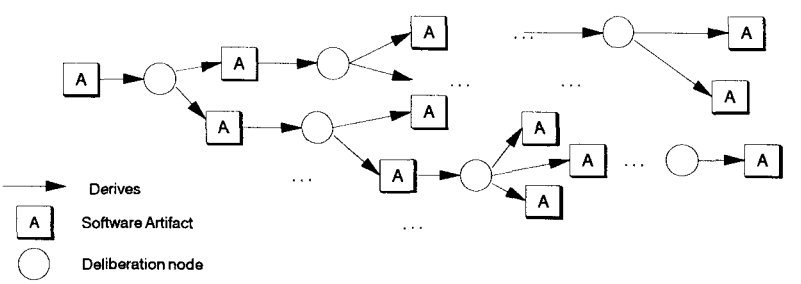

Figure 1. A software evolution graph.

Our central thesis is that a wider perspective is needed in representing reasoning in software development. Supporting only the phases of design and requirements specification, can provide limited help. Reasoning needs to be supported through the whole software life cycle, in a flexible and customizable way (in the sequence, the terms *software evolution* and *software life cycle* will be considered equivalent). Assumptions should be recorded at a global level so that their role can be thoroughly

understood. A Knowledge Base containing knowledge about problem-solving practice in software evolution should also be maintained. PROFILE, introduced in this paper, is a logic-based conceptual meta-model aiming to provide a framework for reasoning support in software evolution. It enhances existing models by providing support for the whole software life cycle, by handling assumptions as global reasoning elements, and by maintaining a Knowledge Base for software evolution.

This paper is organized as follows: in section 2, we briefly present our experience with existing deliberation representation models in capturing software design rationale, and give a detailed description of a new logic-based conceptual meta-model. A prototype system implementing an instance of this meta-model is presented in section 3. In section 4 we discuss the related work and in section 5 we make some concluding remarks and present directions for our future work.

2. A new logic-based approach

2.1. Experience

We have been experimenting with software deliberation representation in the phase of design, using qualitative prototypes of existing data models [Potts88], [Lee90], [Lee91], [Conklin88], [Vesc95]. Our experimentation has been done on the design of three software applications: a small-sized *text formatter* developed for illustrative purposes, a medium-scale *document manager*, and a large-scale *technical construction cost estimation* application. We came out with a few useful observations:

- The representation of software design deliberation is a relatively new topic in software engineering. It is not obvious *what* is useful to be captured and *how* to exploit such information.
- A limitation of the models we used, is that the notion of *Design Artifact* is not well-defined [Potts88] [Lee91]. It is not clear neither what exactly an *Artifact* is, nor what its difference from an *Alternative* is.
- Not all alternatives are compatible to each other: selection of another alternative at a later time, may not be possible because of incompatibility with alternatives selected up to then.
- The notion of "design policy" does not exist in any model we have considered.
- *Assumptions* taken into consideration during design were not represented as such, although they are the bridge between the real world and a software system and their representation in a conceptual model should clearly reflect this fact [Lehman91].

2.2. A hierarchy of reasoning in software evolution

From our experimentation, it has been made clear that there are two levels of hierarchy of practical reasoning in software evolution (Figure 2). The lower one is the *application-specific* level and contains reasoning information which is unique to the software application during the life cycle of which reasoning is supported. Such information includes the deliberation elements during evolution problem-solving and is captured using the *Deliberation Representation Data Model.* The higher level is called *global* and contains reasoning information which is global to a software development organization, or at least, to a specific problem-domain in which many software applications can be developed.

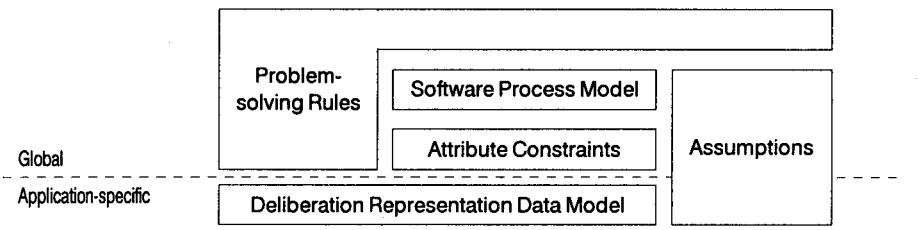

Figure 2. Levels of reasoning in software evolution.

The global level contains the following elements:

The software process model followed, which represents the phases in software evolution, an analysis of each phase in a hierarchy of tasks, and the software artifacts produced in each task. If the definition of the software process model is done at a highly-detailed level, the internal structure of the software artifacts, is also represented.

The constraints that are applied on the evaluation of software artifacts using any measurable attributes, and

A rule base containing knowledge about the implementation of the process model and the practice followed by the developer during evolution problem solving.

Assumptions represent conditions or the real world and in particular of the software development environment or the problem domain, which are true for a period of time and can be used as reasoning elements in deliberation recording. Usually, the scope of an *assumption* is wider than a single evolution problem; however, an *assumption* could also be used as a reasoning element at the application-specific level as shown in Figure 2. With respect to this classification, current approaches provide reasoning support at the application-specific level [Potts88][Lee90][Arango93], and only partially embody the notion of assumption [Ramesh92] as a reasoning element.

2.3. Requirements from a data model

Based on the discussion so far, we present several requirements from a model that will be used to capture deliberations in software evolution.

Software life cycle representation. Software life is divided in discrete phases in which several actions take place. A software evolution model is a set of software engineering activities, methods and practices, needed to produce, enhance, repair, and maintain software [Humphrey89]. Such a model defines what actions should be performed and in what sequence. As these actions take place, software *Artifacts* are produced. Software evolution models do not define the same types of artifacts, or consider similar types of artifacts in an equivalent way. Reference to the software evolution model is necessary for a fine definition of *Artifact*. Hence, the software life cycle followed, should be a part of the model. Apart from a better definition of *Artifact*, this could be useful in classifying decisions, and in understanding the impact and the importance of decisions according to the evolution phase they have been made in.

Development practice representation. Developers use software metrics to define quality measures for the software applications they produce, and require software artifacts to comply to certain norms or constraints when evaluated using these measures. This is a global software development organization's policy that can justify decisions in a proper context, independently of the specific software application in the development of which these decisions have been made. Another case of policy is that of specifying steps that should be taken in solving specific problems. Such requirements can be considered as *decision factors*, because they provide guidance on what piece of information should be considered significant in a problem-solving activity in software evolution. *Decision factors* can therefore be (a) constraints on evaluation, if metrics are well-defined, (b) logic rules, in the case of problem-solving steps, or (c) free-text containing general guidelines, if neither can be formulated. All three types of *decision factors* can be expressed as rules using predicate logic.

User-defined deliberation representation model. In a deliberation, it is not clear what the most useful information to keep is. The different semantic orientation of the models introduced so far, suggests that this information is not the same in all phases of software evolution. The argument that it can be different even among developers, has a standing basis, too. It is therefore required that any deliberation representation schema, not necessarily the same throughout the whole evolution, should become a part of the model under discussion.

Representation of assumptions in a proper context. Assumptions play an important role in software development [Lehman91]. In e-type applications [Lehman80], they can be considered as a bridge between the real world and software. Attaching assumptions in any software engineering artifact, provides a basis for better reasoning in software evolution by capturing important deliberation information in an appropriate context.

2.4. Description of PROFILE

Figure 3 shows the main elements of a conceptual meta-model developed to satisfy the requirements presented in section 2.3, called PROFILE (**PRO**cess modelling and decision **F**actors **I**n software de**L**iberation and **E**volution). This is an abstract view of PROFILE; as will be described later, many elements need to be replaced with a specific data model, in order to have a real implementable model instance. We call this process "model instantiation".

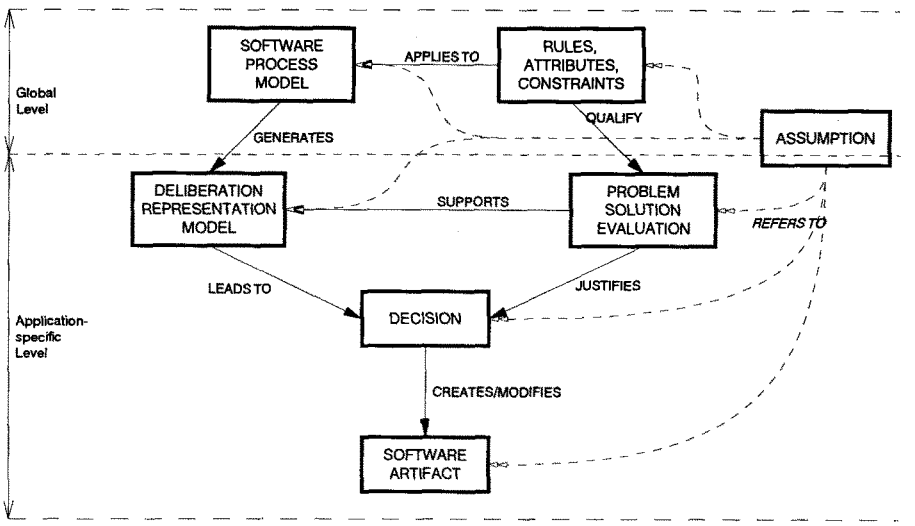

Figure 3. An abstract view of PROFILE.

The first element to define is the software evolution model. This requires representation of phases and their sequence in software evolution. Such a definition can be made in any of three levels of detail as described in [Humphrey89]. In the most detailed level (called the *Atomic* level) a complete definition of all *Artifact* types including their internal structure, is possible.

The next step is the representation of policies followed by developers during software evolution. *Quality measures* is a set of software metrics, formal or not, used by a developer. They usually have a description, a measurement scale, and an applicability scope in terms of *Artifacts* and *Evolution Phases*. A *Decision Factor* is either a constraint upon *Artifacts* evaluation, a sequence of actions taken in problem-solving, or a generic guideline expressed using natural language.

A deliberation representation model for capturing rationale at the *application-specific* reasoning level is the next element of the model to define. It can be the same throughout the whole evolution, or it can be different according to evolution phases. In the latter case, we have a three-dimensional model with reference to the software evolution model used. This implies a higher overhead in software evolution and

should be carefully balanced with expressiveness and exploitation of collected information.

Using *Quality Measures* defined previously, alternative solutions to software evolution problems are qualified by assigning values to attributes of *Artifacts*. This is a key in justifying decisions in software evolution. It is well known that software metrics have failed to provide objective measures of software characteristics, except maybe for program code, but metrics is all we have. Defining *Viewpoints* for supporting multiple qualifications made under different assumptions or by different persons, is a realistic solution to this.

The notion of *Assumption* in PROFILE is rather universal than part of the deliberation representation model. *Assumptions* can be attached to any entity of the model to represent a condition that is true within a time frame. Implementations are responsible for handling the temporal dimension of assumption validity, and determine what happens when an *Assumption* is no longer true.

A *Decision* is a trigger for the creation or modification of software Artifacts. Removal of *Artifacts* is a special case of modification. A fine definition of *Decision* depends upon the deliberation representation model that will be used, and can only be done in an instantiation of PROFILE. A *Decision Signature* is a key notion in PROFILE and is defined as a set containing references to the context in which a *Decision* was made. We thus have: $DSig = \{ DPh, DF^*, Q^*, A^*, J, S^* \}$ where DPh = design phase, DF = applicable decision Factors, Q = solutions qualification, A = standing assumptions, J = justification as recorded in the deliberation representation model, S is the scope of the decision in terms of *Artifacts*, and (*) denotes a set instead of a single element. *DPh*, and *J* are pointers to the corresponding entities of the software evolution and deliberation representation models, respectively. A *Decision* is valid as long as all the elements of its *Signature* are valid. Decision Signatures can be used in submitting queries to the model to extract information about software evolution rationale and will be further discussed in section 2.5.

2.5. Instantiation, implementation and use

In this section we present the actions that need to be taken in order to specify an implementable instance of PROFILE, we address some implementation issues and discuss a few aspects of using such an implementation in a real software development environment. From now on, such an implementation will be called a Software Evolution Decision Support System (SE-DSS). We believe that the data model of a SE-DSS highly depends upon the user characteristics; hence, a SE-DSS cannot be regarded as "one more CASE tool". The use of a SE-DSS implies a measurable overhead, however its necessity cannot be considered well understood by development teams: things could easily end up in a documentation-mess. A developer should proceed in small steps, evaluating experience and using as much feedback as possible from those who actually do the "dirty job" in software development.

At first, the software development organization should specify its requirements from the use of this new dimension of software documentation. The next thing to do, is to conduct some kind of organization-wide research in order to collect some useful characteristics and knowledge that will be embodied in the SE-DSS. These characteristics are: the software life cycle model, the quality measures, the decision factors, and those elements of deliberation in software evolution that will be recorded. Determining the first two, is the easy part of this work. Determining the decision factors is more or less equivalent to creating a special knowledge base by acquiring experts' knowledge on software evolution, which is not an easy thing to do. The initial set of decision factors will contain constraints about metrics and maybe a few guidelines expressed in natural language; as experience is gained, this set will contain fruitful knowledge on software development. It is apparent that a SE-DSS can be best implemented in an environment suitable for knowledge representation. Such a "standard" environment is Prolog, in which we have chosen to implement a prototype presented in section 3; frame-based environments such as ConceptBase [Jarke91] can also be used. In the following discussion we consider Prolog as the implementation environment.

The main functions of a SE-DSS are the maintenance of software evolution history and the relevant information extraction, the maintenance and handling of trigger events whenever certain conditions become true, the maintenance of the knowledge base, and the support of decisions by consulting the knowledge base and examining similar situations of the past. The evolution of a SE-DSS is shown in Figure 4.

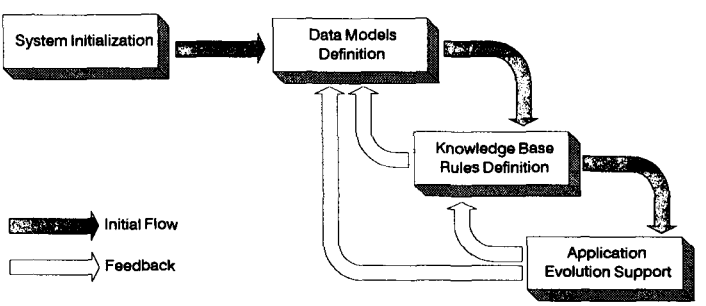

Figure 4. Evolution of a SE-DSS using a PROFILE instance.

3. A prototype SE-DSS system

3.1. Architecture

Figure 5 shows an instance of PROFILE which embodies a deliberation representation model based on DRL [Lee91]. DRL was initially oriented to software design, however, in the context of PROFILE, we think it is generic enough to represent deliberations throughout the whole life cycle.

A prototype SE-DSS system based on this model, called Ictinus*, has been implemented and we have been experimenting with it in the development of several applications. Ictinus's architecture is shown in Figure 6. It is based on the Windows environment where a RDBMS and a Prolog system host a relational database and a rule base respectively. A set of interconnected data models for the representation of the software process model, of the software artifacts and of the deliberation at the application-level, implement the data model shown in Figure 5. A set of Prolog queries implement most of ICTINUS's functions. The main user interface elements are a set of forms for data entry and a graphical browser for browsing through an evolution graph. A front-end user interface provides a working environment to users.

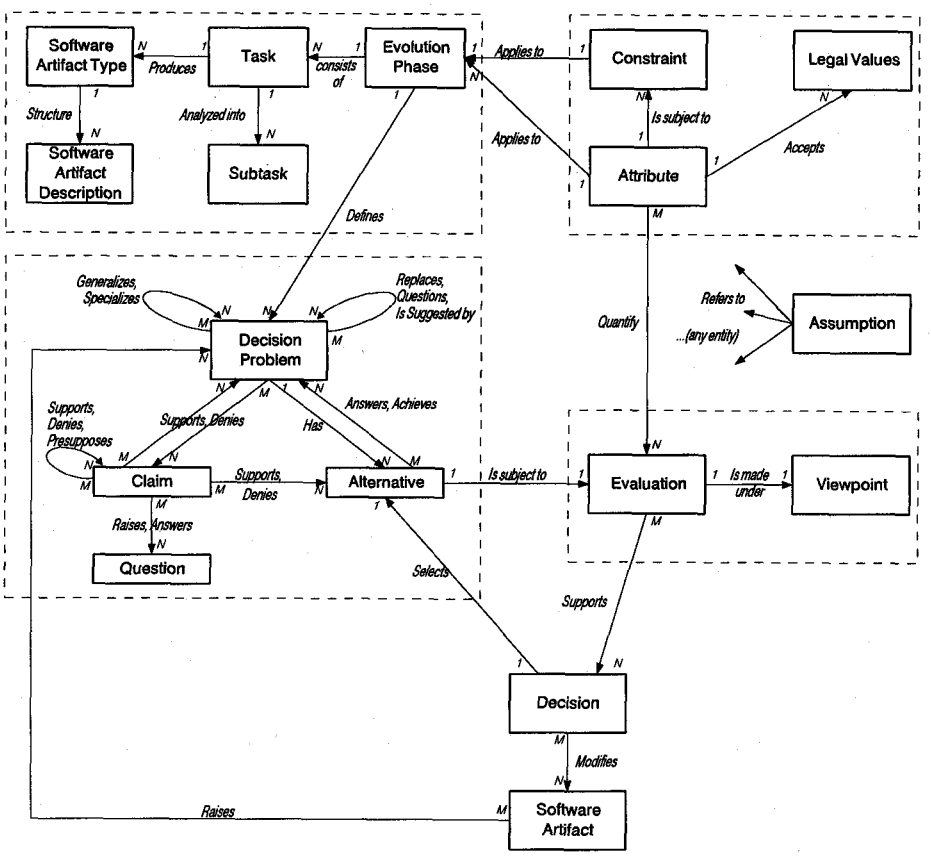

Figure 5. An instance of PROFILE implemented in the Ictinus system.

* *Named after one of the architect designers of Acropolis in Athens.*

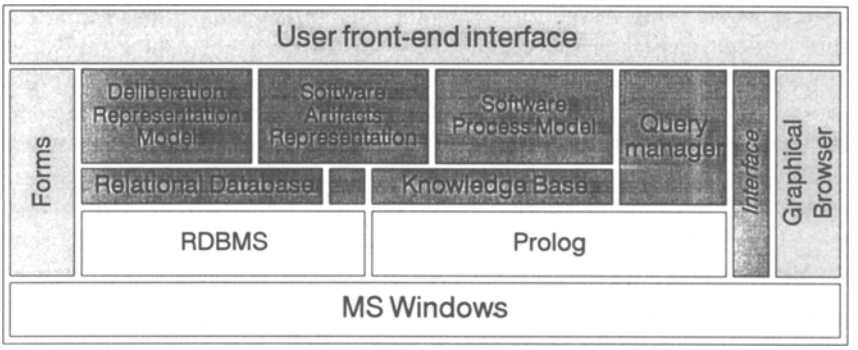

Figure 6. The architecture of the ICTINUS system.

3.2. Services

In the following paragraphs we briefly discuss the main services provided by Ictinus.

Software Process Model management. The user can describe the steps taken in software development at any required level of detail, as a basis for reasoning support during evolution. If this description is done at a high level of detail, the internal structure of software artifacts is also represented and managed. The current implementation of ICTINUS embodies the Waterfall life cycle model at the *Atomic* level of detail, which is the highest with reference to [Humphrey89].

Software Deliberation Representation model management. Several elements of the deliberation representation model (e.g. Viewpoints, Evaluation formulas) reflect the user's policy on evaluation of alternative solution to evolution problems, and therefore are user-defined.

Assumption management. The true state of an assumption is subject to change which results in a set of chain-effects to the decisions in the justification of which this assumption is used. This situation is automatically managed by ICTINUS.

Evolution history recording and playback. Recording of the evolution history is done using either the form-based interface, or the graphical editor. Playback is done using a customizable graphical browser that supports layers of information and can zoom into specific entities to display the required detail. The user decides which classes of entities and what details will be displayed at any time. Walking through the history graph is done using semantic guidance through Prolog queries - not by scrolling a graphical display.

Deliberation management at the application-specific level. Using the deliberation representation data model as a basis, the evolution problems, the alternative solutions and a set of arguments and mathematical evaluations of the alternatives, are stored and managed. Evaluation of alternatives can be made using multiple Viewpoints.

What-if analysis using Assumptions. Modifying current assumptions can affect the validity of decisions taken and change the status of the system. In many cases the system becomes unstable, since some issues cannot be resolved in the new context, and a new problem-solving activity has to take place. Such situations are common during maintenance, where changes in the real world can affect the validity of assumptions and make the system out-of-date, and can be explored using this feature. If some artifacts are modified during this procedure, references to the assumption that initiated the problem are automatically generated and attached to the deliberation information of these artifacts.

What-if analysis using Viewpoints. A similar situation occurs when considering different evaluations of artifacts' quality measures. Depending on the set of decision factors applied, some issues cannot be resolved, which initiates a new problem-solving activity. This is a common situation when doing scenario analysis. Ictinus provides support in such cases by generating alternative evolution graphs that correspond to Viewpoints and by marking unresolved issues. The original evolution graph is modified only when a scenario is accepted and the necessary references to the new viewpoint are automatically generated.

Synthesis library maintenance. A synthesis is a reusable collection of artifacts that share the same qualitative attributes and assumptions in the same functional domain. It can be a set of assumptions, of requirements, of design artifacts, of program elements, or of any type of entities that exist in the software life cycle. A synthesis is *viable* when all attributes and assumptions of its members are valid. Ictinus provides support for maintaining synthesis libraries, for verifying synthesis viability and for performing what-if analysis of the viability of a synthesis in a similar way as in the previous cases.

Decision support is provided through the notion of decision signature. A deliberation takes place in a certain context that is determined by the problem domain, the life cycle phase, the valid assumptions and the applicable decision factors. These elements, except for the problem domain, are part of the *Decision Signature* as defined in section 2.4. We can therefore formulate a query to the knowledge base and extract information about "what has been decided in similar situations". The result of this query can be a long list that can be somehow shortened by running successive refinement queries. The real information obtained from this procedure depends on two factors: The first one is the use of *Assumptions*. Proper use of *Assumptions* can help in effectively characterizing situations and similarities between situations. The second one is the *Decision Factors* rule base. Every time a decision is taken, the user should consider if adding a new rule to the KB is possible. The system's behaviour in decision support is expected to improve as the KB accumulates knowledge about software development practice in the user's organization.

Two other services provided by Ictinus are **event triggering** and **model statistics.** Event triggering is the ability to automatically trigger an event when a condition is met. This is useful whenever a decision becomes invalid for some reason and no other solution exists. The user is notified of the situation and can take the necessary actions.

Finally, using the model statistics feature, the user can determine which elements of the KB are most used. These elements should correspond to policies and situations of the real world, which is an indication that Ictinus is well-tuned to the software development organization.

3.3. Intended use

Figure 7 shows the use of ICTINUS in a software development environment. Chief software engineers are responsible for the initialization of the system by defining the global characteristics of the software development organization, and an initial set of rules and constraints. Software development teams use the system for application-specific deliberation recording and support and provide chief software engineers with feedback regarding the development process itself, and the domain experts with feedback concerning the application domain. Chief software engineers and domain experts are responsible for formulating software practice knowledge and domain knowledge respectively, and updating ICTINUS's knowledge base.

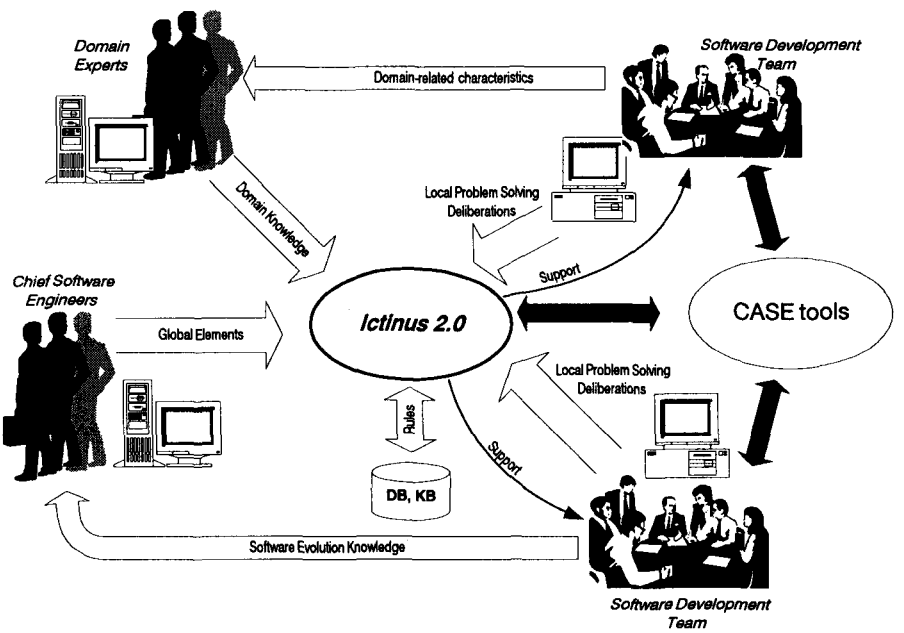

Figure 7. Use of Ictinus in a software development organization.

4. Related work

Potts & Bruns have presented a first approach in capturing deliberations in software design [Potts88]. The data model they proposed is a simple and easy-to-use one. A

justification is a free-text description of the reasons why a decision has been made. Their model cannot capture several arguments on the same issue and relations among them. The software development organization's practice (software evolution model) is not represented, and neither is the problem-solving policy using metrics, constraints and rules. Furthermore, their model is not supported by an integrated tool but instead one has to switch between a hypertext editor and a Prolog environment.

SYBIL is a decision management system [Lee90] implementing DRL (Decision Representation Language) that supports *goals*, *alternatives, claims* and relations among them, in software design deliberation. DRL provides better support for representing argumentation than P&B, and has some common qualitative semantic elements with the IBIS model (DRL's *decision problem* vs. IBIS's *issue*, *alternative* vs. *position*, *claim* vs. *argument*). However, IBIS and its implementations such as gIBIS [Conklin88] and IBE [Lease90] can be considered as generic systems for capturing deliberations in engineering disciplines and do not support explicit representation of goals, or the results of deliberations. Neither of these models supports knowledge acquisition and reuse, or domain characteristics representation. Approaches for domain-oriented design support have been presented by Fisher *et al.* [Fischer92] and Arango *et al.* [Arango93], [Arango93b] however, they are not integrated with deliberation representation schemes, which, we believe, should be done in the future.

REMAP [Ramesh92] is a conceptual model used to capture deliberations during requirements analysis that *relates process knowledge to the objects that are created during the requirements engineering process*. It embodies the IBIS model for capturing argumentation and is supported by a system based on the Telos [Mylop90] language implemented using the ConceptBase [Jarke91] environment. REMAP is useful in *capturing deliberations and knowledge when transforming requirements into design* and enhances previous approaches by adding explicit representation of *decisions*, *constraints* and *assumptions*, and by introducing *process knowledge components*. REMAP does not support all phases of software evolution and the necessity for the *development of a generic model for capturing and representing process knowledge across various phases of systems development* is clearly pointed out by the authors.

The development of such a generic model is the main objective of our approach. The representation of the software process model at a highly detailed level (step hierarchy, products and product internal structure), the maintenance of a software evolution KB, and the ability to use a variable deliberation recording model, are the principal REMAP enhancements that PROFILE suggests. As a consequence, we handle *constraints* and *assumptions* in a different way. *Constraints* are regarded as special cases of *decision factors*, which are rule-based representations of evolution knowledge and are applicable to *evolution phases* instead of to *decisions* and *design objects*. *Assumptions* can be assigned to any entity instead of to *arguments* only, which allows a better understanding of their role in software development. In addition, we provide a set of mechanisms for decision support in software evolution.

Several other models for representing deliberations have been presented [Fischer92] [Dong91] [Siddiqi90] [Rose91] [Toulmin84], but they are either less complete than REMAP, or have a different orientation such as group coordination or domain semantics capturing. Relevant studies also include [Rich92], [Oivo90],

5. Discussion

Reasoning is a new dimension in software documentation. Its necessity is not well understood by developers, and existing data models and tools that support reasoning, seem to be immature and not integrated with other tools used in software development. Although there is much discussion of this topic, a common terminology has not yet been agreed. Furthermore, little has been published about experience and so far, there is not any commercially available CASE tool that supports any of the models that have been presented.

We believe that further research is needed in integrating deliberation representation models with existing methods and tools. A unified approach in using AI techniques in Software Engineering is also useful, especially in determining qualitative similarities among problem-solving contexts. These tasks cannot be accomplished unless enough experience from real software development environments has been gained.

We are currently conducting experiments on the topic of measuring the overhead as well as the long-term gain for developers when capturing deliberations. Such an investigation could provide a basis for a classification of existing deliberation representation models according to a *cost-benefit* ratio. We are also preparing a new series of experiments in commercial software development organizations to support both evaluation of the data model and measurements collection. Experience that will be gained is expected to be valuable for this research.

Bibliography

[Arango93] Arango, Guillermo, and Eric Schoen, and Robert Pettengill, ***A Process for Consolidating and Reusing Design Knowledge***, Proceedings of the 15th International Conference on Software Engineering, IEEE Computer Society Press, pp. 231-242, 1993.

[Arango93b] Arango, G., and E. Shoen, R.Pettengill, and J.Hoskins, ***The Graft-Host Method for Design Evolution***, Proceedings of the 15th International Conference on Software Engineering, IEEE Computer Society Press, 1993.

[Conklin88] Conclin, j., and M.L.Begeman, ***gIBIS: A Hypertext Tool for Exploratory Policy Discussion***, Vol 6, No 4, pp. 303-331, 1988.

[Devanbu91] Devanbu, P., and R. Brachman, P. Selfridge, and B. Ballard, ***LaSSIE: A Knowledge-Based Software Information System***, Communications of the ACM, Vol 34, No 5, pp. 3549, May 1991.

[Dong91] Dong, Jinghuan, and Chris Wild, and Kurt Maly, ***A Software Development and Evolution Model Based on Decision-making***, Proceedings of the 3rd International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Inst., Skokie, IL, USA, pp. 9-14, 1991.

[Fischer92] Fischer, Gerhard, and A. Girgensohn, K. Nakakoji, and David Redmiles, ***Supporting Software Designers with Integrated Domain-Oriented Design Environments***, IEEE Transactions on Software Engineering, Vol 18, No 6, pp. 511-522, June 1992.

[Jarke91] Jarke, M., ***Conceptbase 3.0 user manual***, Univ. Passau, Passau, Germany, Tech. Rep. MIP-9107, March 1991.

[Lease90] Lease, M., and M. Lively, and J. Leggett, ***Using an Issue-based hypertext system to capture the software life-cycle process***, Hypermedia, vol. 2, no. 1, 1990.

[Lee90] Lee, Jintae, ***SIBYL: A Qualitative Decision Management System***, Artificial Intelligence at MIT: Expanding Frontiers, Winston and Shellard, editors, MIT Press, Cambridge, MA, Vol 1, Ch. 5, 1990.

[Lee91] Lee, Jintae, ***Extending the Potts and Bruns Model for Recording Design Rationale***, Proceedings of the 13th International Conference on Software Engineering, IEEE Computer Society Press, pp. 114-125, 1991.

[Lehman80] Lehman, M.M., ***Program life cycles and laws of software evolution***, Proceedings IEEE special issue on Software Engineering, pp.1060-1076, September 1980,

[Lehman91] Lehman, M.M., ***Software Engineering, the software process and their support***, IEE Software Engineering Journal, special issue on Software Environments and Factories, 6(5), pp. 243-258, September 1991

[Mylop90] Mylopoulos, J., and A. Borgida, and M. Jarke, and M. Koubarakis, ***Telos: Representing knowledge about information systems***, ACM Trans. Inform. Syst., vol 8, pp. 325-362, October 1990.

[Oivo90] Oivo, Markku, ***Knowledge-Based Support for Embedded Computer Software Analysis and Design***, Technical Research Centre of Finland, ISBN 951-38-3763-7, September 1990.

[Potts88] Potts, Collin, and Glenn Bruns, ***Recording the Reasons for Design Decisions***, Proceedings of the 10th International

Conference on Software Engineering, IEEE Computer Society Press, pp. 418-427, 1988.

[Ramesh92] Ramesh, B., and V. Dhar, ***Supporting Systems Development by Capturing Deliberations During Requirements Engineering***, IEEE Transactions on Software Engineering, Vol 18, No 6, pp. 498-510, June 1992.

[Rose91] Rose, T., and M. Jarke, and M. Gocek, and M. Maltzahn, and H. Nissen, ***A decision based configuration process environment, Software Eng. J.***, vol. 6 no.5, pp. 332-346, 1991.

[Rich92] Rich, Charles, and Yishai A. Feldman, ***Seven Layers of Knowledge Representation and Reasoning in Support of Software Development***, IEEE Transactions on Software Engineering, Vol 18, No 6, pp. 451-469, June 1992.

[Siddiqi90] Siddiqi, J.I., and J.H.Sumiga, and B. Khazaei, ***Use of a Blackboard Framework to Model Software Design***, Empirical Foundations of Information and Software Science V, Edited by P.Zunde and D.Hocking, Plenum press, NY, pp. 99-107, 1990.

[Toulmin84] Toulmin, S., and R. Rieke, and A. Janik, ***An introduction to reasoning***, 2nd edition, NY Macmillan, 1984

[Vesc95] V.C.Vescoukis, ***Software Design Decisions***, proceedings of CAiSE 95 W3, Jyvaskyla, Finland, June 1995.

[Humphrey89] Humphrey, S. Watt, ***Managing the Software Process***, SEI, Addison-Wesley 1989, ISBN 0-201-18095-2.