IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 3 Issue 5, May 2016.
www.ijiset.com
ISSN 2348 – 7968

Issue-Based Models and Systems in Software Engineering A Survey

Zeynab Rashidi[1]

[1] Department of Mathematics and Computer Science, AmirKabir University of Technology, Tehran, Iran

Abstract

Software engineering is a rationale-driven activity in which engineers are acquiring knowledge and making decisions about the system or its application domain. Software engineers also need to capture the context in which decisions are made and the rationale behind these decisions. Rationale information, represented as a set of issue models, enables software engineers to understand the implication of a proposed change when revisiting a decision. In this paper, we did a survey on four Issue-Based Models in software engineering. These models are (a) IBIS (Issue-Based Information System); (b) DRL (Decision Representation Language); (c) QOC (Questions, Options, and Criteria), and (d) the NFR Framework.

Keywords: *Software Engineering, Issue, Modeling, Rational*

1. Introduction

During software development, Rationale is the justification of decisions 0. Given a decision, its rationale includes the problem that it addresses, the alternatives that developers considered, the criteria that developers used to evaluate the alternatives, the debate developers went through to achieve consensus, and the decision. Rationale is the most important information developers need when changing the system. If a criterion changes, software developers can reevaluate all decisions that depend on this criterion. If a new alternative becomes available, it can be compared with all the other alternatives that were already evaluated. If a decision is questioned, they can recover its rationale to justify it.

There are several development and management methods covering many different aspects of complex and changing software systems. For dealing with system and application domain complexity, there are some methods such as use case modeling, object modeling, architectural styles, reusing design patterns, and mapping of models to source code. For communicating knowledge about the system, software engineers can employ UML diagrams, issue models, informal communication mechanisms, and documents.

Unfortunately, rationale is also the most complex information developers deal with during software development, and thus, the most difficult to update and maintain. To deal with this challenge, software developers capture rationale during meetings and on-line discussions, represent rationale with issue models, and access rationale during changes.

This research focuses on using issue modeling to capture rationale and drive decisions, which can integrate and improve both the technical and social aspects of software development. The structure of remaining parts of this paper is as follows: Section 2 is dedicated to the most important definitions. Section 3 is the survey done on the topic. Section 4 presents the summary and conclusion.

2. Definitions

In this section, we describe several important definitions used in the paper. These definitions are **Issue**, **Issue-Based Information System, sub-issues, consequent issues** and **Issue modeling.**

An **issue** represents a concrete problem, such as a requirement, a design, or a management problem in software development. An issue should focus only on the problem, not on possible alternatives to address it. A convention that encourages this is to phrase issues as questions. To reinforce this concept, we also include a question mark at the end of the issue name. For example in the train scheduling and routing system: (a) *How soon should a dispatcher be notified of a train delay? (b) How should persistent data be stored?* and *(c) Which technology presents the most risk?* Issues most often represent problems that do not have a single correct solution and that cannot be resolved algorithmically. Issues are typically resolved through discussion and negotiation. More information on types of negotiations are given in 0.

Issue-Based Information System (IBIS) is an issue model proposed by Kunz Rittel composed of three types of nodes: issue, position, and argument. Software developers represent issues in UML with instances of class Issue. Issues have a subject attribute, summarizing the issue, a description attribute, describing the issue in more detail and referring to supporting material, and a status attribute, indicating whether the issue has been resolved or not. The status of an issue is **open** if it has not yet been resolved and

ISSN 2348 – 7968

closed otherwise. A closed issue can be reopened. By convention, developers give a short name to each issue, such as 'train delay? : Issue', for reference. For example, Figure 1 depicts the three issues we gave as examples in the train scheduling and routing system.

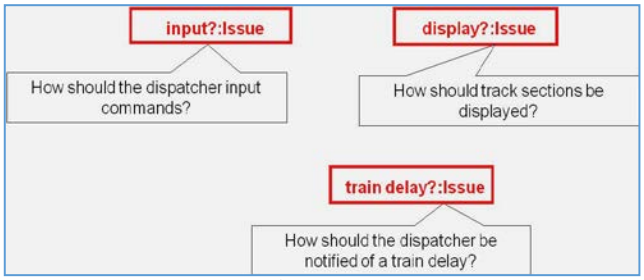

Fig. 1 An example of issues as an UML object diagram 0

Issues raised during software development are often related. Some issues can be decomposed into smaller **sub-issues**. For example the train scheduling and routing system, *"What are the response time requirements of the traffic control system?"* includes *"How soon should a dispatcher be notified of a train delay?"*. The complete system development can be phrased as a single issue—*Which traffic control system should we build?*—that can then be decomposed into numerous sub-issues.

Issues can also be raised by decisions made on other issues. For example, the decision to cache data on a local node raises the issue of maintaining consistency between the central and cached copies of the data. Such issues are called **consequent issues**.

Issue modeling is based on the assumption that design occurs as a dialectic activity during which software developers solve a problem by arguing the pros and cons of different alternatives. Developers can then capture rationale by modeling the argument that lead to the development decisions. In Issue modeling, there are: (a) a question or a design problem as an issue node ; (b) alternative solutions to the problem as proposal nodes; (c) pros and cons of different alternatives using argument nodes; (d) decisions to resolve an issue as a resolution node and (e) implementation of these resolutions as action items.

Applications of rationale to software engineering are few (See 0and 0). Major challenges include the technical issues (e.g., rationale models are large and difficult to search) and non-technical issues, in particular, the perception that rationale is an overhead that benefits only other participants 0. However, there have been successful cases that have illustrated the importance of tightly integrating the capture and use of rationale within a specific process.

During software development, major effort is spent in capturing rationale as decisions are made. Usually, rationale information is documented as a separate model and cross-referenced with other models. For example, issue models represent rationale with a graph of nodes, each representing an issue, an alternative, or an evaluation criteria. The rationale behind the requirements model can then be captured by attaching issue models to use cases.

3. The Survey of the Issue-Based Models and Systems

In 1970, Kunz and Rittel originally proposed the capture of rationale in software engineering as an issue model0. Since then, many different models have been proposed and evaluated for software engineering and other disciplines. In this section, we compare four important Issue-Based Models and Systems. These models are (a) IBIS (Issue-Based Information System); (b) DRL (Decision Representation Language); (c) QOC (Questions, Options, and Criteria), and (d) the NFR Framework.

3.1 The First Model: Issue-Based Information System

The first model is **Issue-Based Information System (IBIS)**. It includes an issue model and a design method for addressing ill-structured or wicked problems (as opposed to tame problems). A *wicked* problem is defined as a problem that cannot be solved algorithmically but, rather, it has to be resolved through discussion and debate.

The IBIS issue model, illustrated in Figure 2, has three nodes (Issues, Positions, and Arguments) related by seven kinds of links (supports, objects-to, replaces, responds-to, generalizes, questions, and suggests). Each Issue describes a design problem under consideration. Software developers propose solutions to the problem by creating Position nodes. While alternatives are being generated, software developers argue about their value with Argument nodes. Arguments can either support a Position or object-to a Position. Note that the same node can apply to multiple positions. The IBIS model did not originally include Criterion and Resolution.

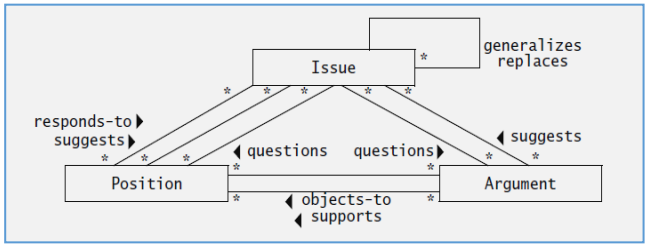

Fig. 2 The IBIS model as an UML class diagram 0

IBIS is supported by a hypertext tool, which is called gIBIS. This tools is developed by Conklin & Burgess-Yakemovic 0 and used for capturing rationale during face-to-face meetings. It provided the basis for most of the subsequent issue models, including DRL and QOC. This

ISSN 2348 – 7968

tool is used for an extensive case study in industry. It has led to a commercial tool called *QuestMap*, which has been used effectively in the context of capturing and structuring rationale during meetings.

3.2. The Second Model: Decision Representation Language

The second model is **Decision Representation Language (DRL).** It aims at capturing the **decision rationale** of a design 0. A decision rationale is defined as the representation of the qualitative elements of decision making, including the alternatives being considered, their evaluation, the arguments that led to these evaluations, and the criteria used in these evaluations. DRL is supported by SYBIL and it is a tool that enables the user to track dependencies among elements of the rationale when revising evaluations. DRL elaborates on the original IBIS model by adding nodes to capture Design Goals and Procedures. DRL views the construction of the rationale as a task comparable to the design of the artifact itself. DRL is summarized in Figure 3. The main drawbacks of DRL are its complexity (7 types of nodes and 15 types of links) and the effort spent in structuring the captured rationale.

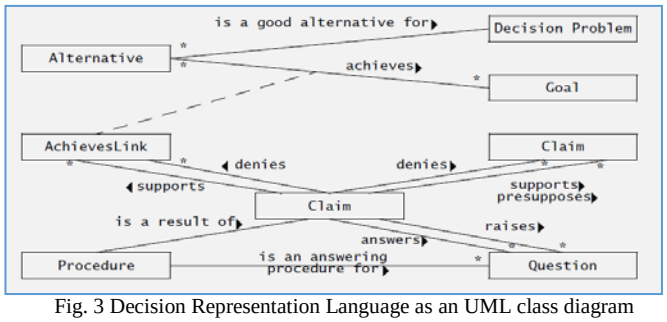

Fig. 3 Decision Representation Language as an UML class diagram

3.3 The Third Model: Questions, Options, and Criteria

In the early 1990s, MacLean et al. proposed an alternative model to IBIS 0, focusing on the systematic evaluation of options against criteria. It is opposed to the opportunistic posting of arguments. In this model, there are Questions, Options, and Criteria (QOC). In fact, it is another elaboration of IBIS. Questions represent design problems to be solved (Issues in the issue model that presented). Options are possible answers to Questions (Proposals in the model 0). Options can trigger other Consequent Questions. Usually, options are assessed negatively and positively against Criteria, which are relative measures of goodness defined by the software developers. Also, arguments can support or challenge any Question, Option, Criterion, or relationship between those. Arguments may also support and challenge Arguments. Figure 4 depicts the QOC model.

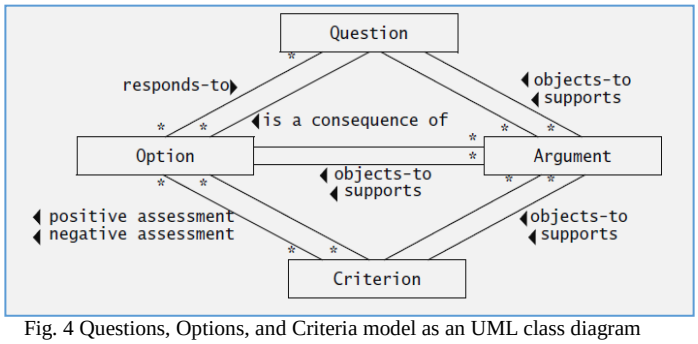

Fig. 4 Questions, Options, and Criteria model as an UML class diagram

To compare QOC and IBIS, we can express that IBIS is a more appropriate representation for capturing rationale on the fly, whereas QOC is a more appropriate representation for structuring rationale for long-term use. Moreover, QOC and IBIS differ at the process level. IBIS's aim, on the one hand, is to capture design argumentation as it occurs (e.g., gIBIS is used for capturing information generated during design meetings). QOC structures, on the other hand, are constructed as an act of reflection on the current state of the design. This conceptual separation of the construction and argumentation phases of the design process emphasizes the systematic elaboration and structuring of rationale as opposed to capturing it as a side effect of deliberation. Rationale, from QOC's perspective, is a description of the design space explored by the software developers. From IBIS's perspective, rationale is a historical record of the analysis leading to a specific design. In practice, both approaches can be applied to capture sufficient rationale.

3.4 The Fourth Model: The Non-Functional Requirements (NFR) Framework

The **NFR Framework** is a method for tracking the relevant nonfunctional requirements for each decision, evaluated alternative, and interaction between nonfunctional requirements **0**. Unlike the previous three issue models, the NFR Framework is specific to requirements engineering. Nonfunctional requirements are treated as *goals* to be met.

To address the difficulty that nonfunctional requirements are usually high-level and subjective, goals are refined and clarified by decomposing them into sub-goals. In the goal graph, goals and sub-goals are represented as nodes. Decomposition relationships are represented as directed arcs. The NFR Framework provides two types of decompositions:

- *AND decomposition:* A goal can be decomposed into sub-goals, all of which need to be met to help the parent goal.

ISSN 2348 – 7968

- *OR decomposition:* A goal can be decomposed into alternative sub-goals, any one of which needs to be satisfied to help the parent goal.

The top-level goals (specified by the client and the users) are hence refined into lower level and more concrete goals. Note that a single sub-goal can be related to more than one parent goal. Moreover, the NFR Framework provides additional types of links to capture other relationships. For example, correlation links between two goals indicate how one goal in the graph can support or hinder the other goal. Since nonfunctional requirements are rarely qualities that are either met or not, links in a goal graph represent how much a goal contributes to or hinders another goal. A goal is *satisficed* (as opposed to *satisfied*) when the selected alternative meets the goal within acceptable limits. Otherwise, the goal is said to be *denied*. Five weights can be associated with a link: *makes*, *helps*, *neutral*, *hurts*, and *breaks*. Root nodes represent high-level goals specified by the client. As these goals are successively refined into more concrete ones, the refinement activity moves towards system features. Goals that represent system features (as opposed to nonfunctional requirements) are called *operationalizing goals*.

Figure 5 depicts a partial goal graph indicating alternatives for an ATM authentication mechanism. In this goal graph, Flexibility, Low cost, and Security are high-level goals. Security is then decomposed using an AND relationship into Authentication, Confidentiality, and Integrity, denoting that for an account to be secure, the system must allow only authorized users to operate on the account, that all transactions must be held confidential, and that all transactions yield a valid result (i.e., no money can be created or destroyed by abusing the system). The Authentication sub-goal is further refined, using an OR relationship, into 'Account+PIN', 'SmartCard+PIN', and 'FingerPrint' reader. These last three goals, since they represent different features of the system, are operational goals, and therefore, are represented using thicker circles. There are two correlation links to indicate (for example) that the 'Account+PIN' goal helps the low cost goal but hurts the Flexibility goal.

Once an initial goal graph has been refined and several operational goals have been developed, different subsets of operational goals can be evaluated and selected. By following the decomposition and correlation links, the software developers can check whether the high level goals are met or they are at least to a sufficiently reasonable extent. In Figure 4, we evaluate the 'Account+PIN' goal as a solution and observe that the Authentication and Low cost goals are satisfied, that the Security goal will be satisfied once the Confidentiality and the Integrity goals are satisfied, but that the Flexibility goal is not satisfied. For realistic problems, the goal graph becomes substantially larger, at which point tool support would be necessary for evaluating and comparing different alternatives. Other aspects of the NFR Framework not covered here include the reuse of goal decompositions from one system to another. This enables the developer to pick groups of solutions that were previously explored and evaluate them against new criteria.

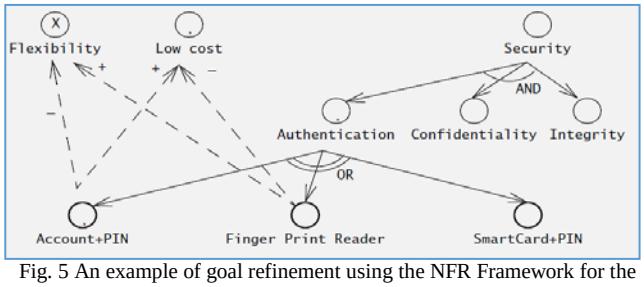

Fig. 5 An example of goal refinement using the NFR Framework for the ATM authentication mechanism (NFR goal graph)

Several collaboration tools enable users to attach weights to different arguments in the issue model, and thus to compute which proposal should be selected with an arithmetic formula 0. The WinWin tool 0uses an issue model similar to the QOC model (See Figure 6). A significant difference is that each node in the issue model is attached to a taxonomy category that represents the application domain under consideration. The Criteria node is named Win Condition and represents a stakeholder success criterion. The WinWin process starts by identifying key stakeholders and their Win Conditions. Conflicting Win Conditions are detected and discussed using the issue model. Once an Agreement is achieved, it is entered in the tool and linked back to the conflicting Win Conditions. This ensures that consensus is found and documented.

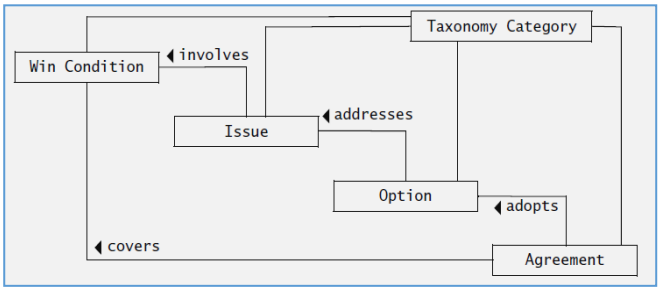

Fig. 6 the WinWin issue model as an UML class diagram

4. Summary and Conclusion

In this paper, we did a survey on four Issue-Based Models in software engineering. These models are (a) IBIS (Issue-Based Information System); (b) DRL (Decision Representation Language); (c) QOC (Questions, Options, and Criteria), and (d) the NFR Framework. These models are very useful, when we use issue modeling to represent the information exchanged during these negotiations to

ISSN 2348 – 7968

help capture rationale. We can also use issue modeling to facilitate negotiations. Adding a resolution to an issue model effectively concludes the discussion of the corresponding issue. As software development is iterative, it is sometimes necessary to reopen an issue and reevaluate competing alternatives. At the end of development, however, most issues should be closed or listed as known problems in the documentation.

References

[1] A. H. Dutoit & B. Paech, "Rationale management in software engineering," in S.K. Chang (ed.), *Handbook of Software Engineering and Knowledge Engineering*, Vol. 1, World Scientific Publishing, 2001.
[2] A. H. Dutoit & B. Paech, "Rationale-based use case specification," *Requirements Engineering Journal*, 7(1): 3–19, 2002.
[3] A. H. Dutoit, R. McCall, I. Mistrik, B. Paech (eds.) *Rationale Management in Software Engineering*, Springer, Heidelberg, 2006.
[4] A. MacLean, R. M. Young, V. Bellotti, & T. Moran, "Questions, options, and criteria: Elements of design space analysis," *Human-Computer Interaction*, Vol. 6, pp. 201–250, 1991.
[5] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, & R. Madachy. "Using the WinWin spiral model: A case study," *IEEE Computer* 31(7): 33–44, 1998.
[6] B. Bruegge and A.H Dutoit, "Object-Oriented Software Engineering: Using UML, Patterns, and Java", Pearson Prentice Hall, 2010
[7] J. Conklin & K. C. Burgess-Yakemovic, "A process-oriented approach to design rationale," *Human-Computer Interaction*, Vol. 6, pp. 357–391, 1991.
[8] J. Lee, "A qualitative decision management system," in P. H. Winston & S. Shellard (eds.), *Artificial Intelligence at MIT: Expanding Frontiers*, Vol. 1, pp. 104–133, MIT Press, Cambridge, MA, 1990.
[9] L. Chung, B. A. Nixon, E. Yu, & J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic, Boston, 1999.
[10] M. Purvis, M. Purvis, & P. Jones, "A group collaboration tool for software Engineering projects," *Conference proceedings of Software Engineering: Education and Practice (SEEP'96)*, Dunedin, NZ, January 1996.
[11] R. Fisher, W. Ury, & B. Patton, *Getting to Yes: Negotiating Agreement Without Giving In*, 2nd ed., Penguin Books, New York, 1991.
[12] T. P. Moran & J. M. Carroll (eds.), *Design Rationale: Concepts, Techniques, and Use*, Lawrence Erlbaum Associates, Mahwah, NJ, 1996.
[13] W. Kunz & H. Rittel, "Issues as elements of information systems," *Working Paper No. 131*, Institut für Grundlagen der Plannung, University Stuttgart, Germany, 1970.