A Framework For Capturing Design Rationale Using Granularity Hierarchies

Vikas Arora Jim E. Greer J. Paul Tremblay
Department of Computational Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada - S7N 0W0

Abstract

Software design is the central unifying activity of software engineering. It is highly complex, knowledge intensive and pervasive in the software development life-cycle. There are a large number of design methods available varying in terms of their analytic power and the richness of information they provide. These methods are weak in dealing with the incremental nature of design and do not provide support for intuition and judgement on the part of the designer during the design process. A framework is required to capture design rationale, i.e., the decisions made during the design process and the reasons as to why they were made. Capturing design rationale will enable a system to monitor the design process and hence provide intelligent assistance. Two issues related with capturing design rationale are, 'the sequence of design decisions made', and 'how to represent this sequence of decisions'.

1 Introduction

Designing software is an evolutionary process. The goal of a designer is to evolve a representation or a model of an entity which can be built later. The design process involves intuition and judgement based on experience. The quality of design improves as a broader understanding and a better analysis of the system takes place. The foundations of software design seem to be relatively weak; software design methodologies seem to lack depth and flexibility. There are a large number of design methods available, varying in terms of their analytic power and the richness of information they provide. These methods are weak in dealing with the incremental nature of design and provide little support for intuition and judgement on the part of the designer during the design process. With the growing size of software projects, one person cannot have all the knowledge to support and develop these systems and usually a large team of developers is involved. Today the trend is toward the development of intelligent tools and interfaces, not only to ease software development but also to enhance the productivity of software developers[4][8][11].

The goal of our research is to evolve a 'Generic Design Assistant' (GDA) which will have a capability to monitor the designer/maintainer(s) performing design and to provide appropriate assistance based on the situation. We have identified the different components of GDA (see figure 1) and in this paper we look at one of the important components of the GDA, the 'Design Rationale System' (DRS). We propose a framework for capturing and representing design rationale by extending the Potts Model[10] and using a knowledge representation scheme known as the granularity hierarchy[7]. The objective is to provide a formal representation for design-process artifacts which is not only capable of representing design process information but also takes care of the rationale which evolves as the design unfolds.

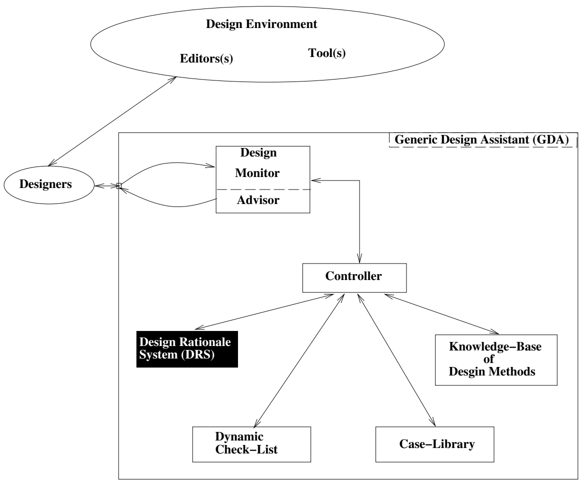

Figure 1: A Design Environment with GDA

In this environment, the designer, while designing a system using any semi-formal design methods, will be required to simultaneously pass on the rationale behind taking various design decisions with respect to problem decomposition, control transfer, data transformation and data co-ordination to the DRS. The DRS employs the Potts Model for encapsulating design information and our extensions encapsulate the rationale behind taking those decisions. The rationale which is captured is represented in a granularity hierarchy of design-process artifacts. This rationale is in the form of design decision snapshots pertaining to the state the designer is in. For example, if the designer is evolving a design artifact, the snapshots pertain to various issues which arise and what is done to resolve them. All the design-process artifacts are arrayed in the granularity hierarchy in such a manner that at any instant of time the hierarchy of design-process artifacts gives the overall view of the design evolution. It represents the artifacts in the form of a graphical structure, where each node of the graph is a design-process artifact and the link is the derivation path. Here the coarser grain design-process artifacts lie at the top of the hierarchy, whereas the finer grain artifacts appear lower in the hierarchy. Thus, at any instant of time during the design, the DRS will be able to provide information regarding how the designed has evolved.

The design-process artifacts in the DRS are evolved along with the design-product artifacts in the design environment. For example, if a designer is using the 'Structured Design' paradigm for evolving the design-product artifacts, the DRS assists in creating design-process artifacts that correspond to these product artifacts.

In the next Section we look at the design rationale component. Section 3 looks at the Potts Model and its limitations, Section 4 proposes extensions to the Potts Model. A schematic notation to the Extended Potts Model is also presented. Section 5, looks at the knowledge representation aspects of the design rationale by introducing granularity hierarchy and applying it for the representational purpose in Section 6.

2 Design Rationale

Design rationale looks at decisions made during the design process and the reasons as to why those decisions were taken, i.e., how many artifacts have been created, what was the reason behind their creation, what alternatives existed, which one was selected and why, etc. A detailed survey on design rationale from a software engineering perspective appears in [5]. It looks at the different models used and the systems that have evolved.

Capturing design rationale will help in recording the design decisions and the reasoning behind the design, which can be later analyzed to control the overall design and manage the complexity of the design process. This will also have its impact on maintenance, which has been observed to be very much dependent on the design. Capturing design rationale and providing a suitable representation scheme will also help in reverse engineering of the design and creating libraries of design-process artifacts for design reuse and design traceability. Furthermore, the rationale can be used to reason about the changes made by the designer based on the knowledge base generated. The point which is emphasized here is that, if design rationale can be captured, a contribution can be made towards intelligent assistance by validating the design goals against design actions and the rationale behind generating the design-product artifacts.

Capturing design rationale requires a data model for encapsulating design decision and analysis information as the design evolves. Furthermore, the model should be generic enough to model processes associated with various semi-formal methods (JSD, Structured Design, Object-Oriented, Higher Order Software (HOS), Box Notations, etc.). For dealing with design rationale, we look at the Potts generic model for design process modelling[10].

3 The Potts Model

Potts defines a generic schema based on an entity-relationship-attribute data model to describe design process (see figure 2) [1]. The model has five entities: artifact, step, position, issue, and argument. An artifact represents information about a design state. A step is basically a design step used for bringing forth refinements and revisions and is defined as an operation which creates or modifies artifacts. The three other entities, issues, positions and arguments, make explicit the reasoning behind design decisions. This particular notion of representing design decisions in terms of issues, positions and arguments stems from Rittel's work [12] in using his IBIS (Issue-Based Information System) model to support the discussion of policy and design alternative in architectural planning.

Based on Potts' five entities, the evolution of software design can be put in the form of the following algorithm:

```
1- Problem statement.
2- State requirement specifications for
   the problem.
3- Repeat
     Get an Artifact
      Repeat

      Initiate a 'Step' => Raise an 'Issue'
                OR/AND  Modify an 'Artifact'

      Raise an 'Issue' => Review 'Artifact'

      State a 'Position' => Respond to 'Issue'
               OR/AND Contribute to a 'Step'

      State 'Arguments' => Support/Object
                           to a 'Position'

      Until (Artifact is resolved)
   Until (Whole design is resolved)
```

In this algorithm, generating an artifact involves performing a number of design steps which leads to raising a number of design issues. Thus initiating a step will lead to raising an issue and hence modifying an artifact. An issue can also simply review an artifact. There will be a number of positions responding to an issue, which may or may not contribute to a step. Arguments provide evidence to artifacts and each position will be supported or objected to by a number of arguments. Such a process need not be done in a sequential way.

Apart from the information being represented by the above design algorithm, there is a great deal of

[1] The entities and the relationships linked by dotted lines in figure 2 are part of the extensions to the Potts Model.

information created during the evolution of design-process artifacts which the existing Potts Model is not able to capture. This information includes answers to questions such as:

- Why was the change done?
- When was the change done?
- Who did the change?
- What change was done (addition/ deletion/ clarification/ deliberation)?
- What was the priority of the change?
- What is the estimated impact of the change?

Such information is not only of great significance for analyzing the design states, but also is of great use for the use/reuse of the designproduct/process artifacts, reverse engineering of design and design maintainability. However, it is not only important to capture design rationale on the artifacts generated but also to show the linkage between these artifacts. This model also falls short of providing/supporting any linkages (in an explicit manner) between the various design-process artifacts which get generated as a result of steps taken on the parent artifacts. For example, if we consider the generation of 'Data-Flow-Diagrams', we know that we can move from a level-1 DFD to level-3 DFD and at each level the amount of information generated is quite different. The existing Potts Model is not able to encapsulate and show relationships between them. To deal with the above listed limitations, we propose extensions to the existing Potts Model.

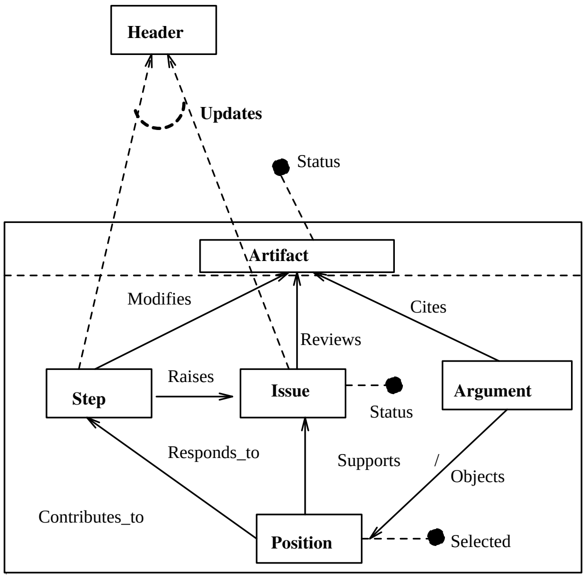

Figure 2: Design Process Artifact

4 Extensions to the Potts Model

Figure 2 shows a design process artifact in the Extended Potts Model. It has a new entity named *header* which is linked to *Step* and *Issue* by a binary relation*updates*. The relation is binary as at any instant of time either *Step* or *Issue* updates the *header*. The entity *Step* updates header when it modifies an artifact, i.e., clarification or deliberation to an already existing *Issue*. The entity *Artifact* now has an attribute named *Status* which can have a value as *Active* or *Processed*, depending upon whether the artifact has been fully resolved (processed) or is still evolving (active). The entity *Issue* also has an attribute *Status*, which can have values of *Pending*, *Processed* or *Ignored*.

The Header plays a very important role in capturing the design rationale. It holds all the information regarding *Who, What, When* and *Why* a particular action was taken on a design artifact. It has entities like *Time-Stamp*, *Change-Category*, *Priority-Level*, and *Supporting-Artifacts*. Each of these entities have further elaborations, i.e., *Change-Category* goes into *Change-Request*, *Clarification* and *Deliberations*. This captures information regarding any change done on an *Artifact*. The information regarding when a particular change was done (*Time-Stamp*), what was the priority level when the change was done (*Priority-Level*) and what are the supporting artifacts for the current artifact (*Supporting-Artifacts*) is also captured.

The header has the following information structure:

* Header Statement $\Rightarrow$ Time-Stamp $\wedge$ Change-Category $\wedge$ Priority-Level $\wedge$ Supporting-Artifacts
* Time-Stamp $\Rightarrow$ date-time $\wedge$ person-associated
* Change-Category $\Rightarrow$ Change-Request $\vee$ Clarification $\vee$ Deliberation
* Change-Request $\Rightarrow$ reason $\wedge$ estimated-impact $\wedge$ suggested-solution $\wedge$ side-effects
* Clarification $\Rightarrow$ description
* Deliberation $\Rightarrow$ description $\wedge$ reason
* Priority-Level $\Rightarrow$ low $\vee$ medium $\vee$ high
* Supporting-Artifacts $\Rightarrow$ artifact-names

This is header structure is based on the concept of communication sheets [9] which have been used for representing the process models. Thus the *header* contains fields which capture the information related to the questions listed in the previous section. In a step towards formalization for a better control on the model in terms of implementation, verification and validation, we specify the 'Extended Potts Model' through a set of schemata which includes a set of *entity types*, *relationship types* and *attributes* based on an EARA based meta-model proposed by Sorenson et. al[13].

4.1 Schematic Representation for Extended Potts Model

A schema is denoted by a 4-tuple (ET, RT, AN, att) where:

- ET: is a finite set of entity types;
- RT: is a finite set of relationship types;
- AN: is a finite set of attribute names;

- att: is a mapping from an element of the set of entity types to an element of the set of attribute names.

Based on the meta-model the 'Extended Potts Model' can be summarized as:

- ET = {Design-Artifact, Issue, Step, Argument, Position, Header}
- RT = {raises = (Step, Issue), contributes_to = (Position, Step) modifies = (Step, Design-Artifact), updates = (Step, Header) support_to = (Argument, Position), objects_to = (Argument, Position), cites = (Argument, Design-Artifact), responds_to = (Position, Issue), reviews = (Step, Issue)}
- AN = {selected, pending, ignored, active, processed}
- att = {Design-Artifact -> active, processed; Issue -> pending, ignored, processed; Position -> selected}

A detailed EARA meta-model based representation of the Extended Potts Model is presented in [1].

Having been able to capture snapshots of the design rationale with the 'Extended Potts Model', we now consider the problem of representing the evolving set of design-process artifacts. The representation scheme should be able to meet the requirements of the thought process which goes behind the design process, i.e., it should be able to show the linkages, dependencies and constraints between different design-process artifacts and also provide a facility to add, delete, modify and retrieve design decision and analysis information.

Thus to support the incremental nature of the design process, we need a representation scheme which can support features like generalization/specialization, aggregation, membership, modularization, representation abstraction and classification. A 'Granularity Hierarchy' is a knowledge representation scheme which satisfies all the above listed requirements. This scheme has been successfully used in the intelligent tutoring domain to monitor a user performing a task or learning a concept, to perform plan recognition on the observations, and to provide advice[2]. We use 'Granularity Hierarchy' to represent the design decision and analysis information by creating a network of the 'Extended Potts Model' snapshots of design rationale. In the next section, we discuss how granularity hierarchies satisfy the above listed requirements.

5 Granularity Hierarchy

Granularity deals with the level of detail associated with a viewpoint on the world [3]. A concept can be viewed at a detailed level or at a general level or somewhere in between. It can also be viewed as a whole or decomposed into its component parts. The main emphasis of granularity is on *shift of focus* as one proceeds with the task. The power of granularity in representation and reasoning was expressed first by Hobbs[3]. McCalla et.al[7] have given it a more formal framework for use in monitoring learners in the intelligent tutoring domain.

A granularity hierarchy is a semantic-network style knowledge representation scheme. It consists of three types of objects: the S-object, the K-cluster and the Observer, where the S-object represents the domain object known to the system, while K-cluster and observer objects are intermediate structures used for examining and retaining information relevant to S-objects in the hierarchy. Granularity provides two relations, i.e., *abstraction* and *aggregation*. Aggregation (Ag) relates S-objects (coarser-grained) with their (finer-grained) component parts. Abstraction (Ab) relates more approximate or general (coarse-grained) views of an S-object with more detailed (fine-grained) versions of the object. These relations bear some similarity to Part_of and Is_a, the familiar semantic network relations. The Ag and Ab relations can be viewed as two orthogonal granularity dimensions: granularity in the level of detail in viewpoint (abstraction), and granularity in the size of component parts deemed relevant (aggregation). The operations of granularity involve *articulating* from coarser to finer-grained objects (in either dimension) and *simplifying* from finer to coarser-grained objects. Observers interact with the environment and are used when granularity hierarchy is used for recognition. We do not require observers in this application.

As stated earlier, granularity hierarchies support the design process knowledge requirements, lets look at them from granularity viewpoint:

- It supports aggregation by allowing representation of relations between a parent node and its component parts.
- It supports 'Generalization/Specialization' by allowing refinements to be made to a node, i.e., allowing knowledge to be represented in the form of coarser and finer levels of detail. It also provides articulation and simplification functions to move along the hierarchy.
- Granularity supports both 'Membership' and 'Classification' in the form of K-Clusters. The role of a K-Cluster is to encapsulate entities which support its parent class/node according to their common properties along the aggregation dimension. It also supports the concept of having more than one K-Cluster supporting its parent node thereby supporting the member_of relation.
- Modularization is inherent in the concept of granularity, as the nodes which are evolved highlight there own identity in the form of knowledge encapsulated in K-Clusters.
- The concept of 'Representation Abstraction' is also inherent in the organizational feature of granularity. As we move from one level of detail to another, we deal with an altogether different set of requirements (more detailed ones). Thus supporting both 'focus shift' and 'delay in detail'.

6 Methodology

Each artifact which evolves out of the design problem is encoded as an S-object in a granularity hierarchy. A design typically comprises many design-process artifacts related by aggregations and abstractions. Supporting each design-process artifact are five entities based on the Potts Model (see figure 2). These five entities within each design-process artifact are related to one another by binary relations and they capture the evolutionary process of the artifact.

New artifacts can be added to the hierarchy as refinements via the abstraction/ refinement relation or as parts via the aggregation/ part_of relation. Component parts of design-process artifacts (themselves design-process artifacts) are clustered in the aggregation dimension via K-clusters. In this way, as the design process evolves, the hierarchy of design-process artifacts grows. The design rationale of any particular artifact in the hierarchy can be retrieved, inspected, or modified; and through the granularity relations, propagation of effects to related design-process artifacts can occur.

Each artifact maintains a header which has the information as discussed in Section 4.

There is a facility for the designer to add/ delete/ modify the various entities in the artifact. The designer is able to obtain desired information by placing constraints (in the form of a query) on the hierarchy of design-process artifacts.

Whenever a designer issues a query to access or change any information an "Event" is triggered. An event, in our context, can be initiated from the outside world(ex_Event) or generated from within the system(in_Event).

Thus an event can be represented by:

Event = ex_Event | in_Event

ex_Event = $< event\_id > < event\_var\_list >$
$< artifact - S - object >$

in_Event = $< event\_id > < event\_var\_list >$
$< artifact - S - object >$

Queries issued by the designer trigger ex_Events, while queries generated by the system trigger an in_Event. Each event is assigned a unique event-id to keep track of the individual designer. An event will also require a list of variables for query creation and the artifact-S-object on which one wishes to operate.

The granularity hierarchy maintains a view of the design evolution with respect to the current artifact. This feature presents a granularity-based propagation of the changes made by designer on the current artifact. The articulation and simplification functions allow constraint propagation in the hierarchy in the following manner. When a designer is focussed on any artifact in the hierarchy and wants to make a change, the articulation function informs the designer about the dependent objects (the parts forming the K-cluster, or the refinements). Similarly, the simplification function allows the designer to move up the hierarchy from finer grained to coarser grained artifacts. In this way, when a designer edits or considers an artifact, these two functions trigger in_Events to be initiated, leading to a control transfer to the related artifact, ensuring that dependencies are considered with respect to this new artifact. The designer, thus, is guided in traversing the relevant parts of the hierarchy and, in a principled way, manages the effects of design evolution.

Let us see briefly how this framework works with respect to designing a text formatter [6] (see figure 3). Here the problem forms the root node of the hierarchy and is therefore a design-process artifact. As has been said earlier, each artifact has certain issues, arguments, steps and positions associated with it. Here the issue is the problem statement itself, i.e., "*To design a text formatter whose input is a sequence of unformatted text lines mixed with command lines. Where each line (except for the last line) is terminated by a new line character and command lines begin with a period to distinguish them from text lines*". The steps here are read_input, interpret_input and produce_output. There can be a number of arguments and positions associated with the issues and steps. These positions and arguments are raised by the individual designer and are captured in the artifact. This artifact maintains a header and view of the dependencies while the header keeps information regarding "who has done what on the artifact". If a designer wants to move ahead in the hierarchy, e.g., to read_input, this initiates an internal event as read_input in fact refines down to a new artifact, Do_Line, whose main issue is to deal with input lines. This artifact maintains a view which is pertinent from the current position. The issues here would be "Process a block at a time", "Process a line at a time" etc. This artifact is in turn supported by Do_text_line and Do_command_line objects which classify the lines as text or command line. This way a designer can move from one artifact to another in the hierarchy and can view the changes made by other designers/maintainers on the artifact(s). To fetch the information from the artifacts, a query like "Display - All Issues by 'person_X' AND All Issues by 'person_Y"' will be fired and the relevant information will be retrieved. This framework thus helps in dealing with changes/modifications made in the design. In this way, as the design of the text-formatter progresses, the design-process artifact hierarchy also evolves.

A prototype DRS is being implemented based on the Extended Potts Model using granularity hierarchy. The DRS provides a facility to capture the design information in terms of issues, arguments, positions etc. It also captures the motivation behind raising an issue or bringing in a change. A facility to dynamically evolve a design-process artifact hierarchy has been provided, i.e., a designer can attach the newly created artifact anywhere in the hierarchy by declaring any of the existing design-process artifacts as its abstraction and aggregation parents. A facility to cluster the artifacts based on constraints, execution semantics etc. has also been provided. A designer can iterate on any design-process artifact and all the versions which are created are indexed temporally on the same hierarchy node. Several queries have been provided to fetch the information from the design artifact hierarchy. A facility also exists to get information about the immediate parents and children of any design artifact

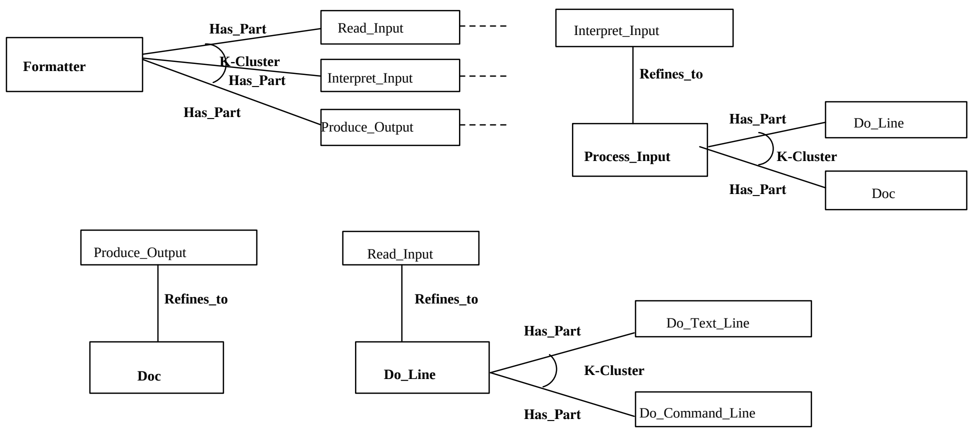

Figure 3: Formatter Design Artifacts in a Granularity Hierarchy

in the design-process artifact hierarchy.

7 Concluding Remarks

We have proposed a framework and built aprototype system for capturing the design rationale using granularity hierarchies. The system shows the feasibility of the concept but still requires user testing to evaluate its usability and expressiveness. Since our long term objective is to evolve a 'Generic Design Assistant' which will have a capability of monitoring the designer and providing advice based on the designers moves, the Design Rationale System is only a first step. We have begun to look at other modules such as the dynamic check-list component and case library.

Acknowledgements

We would like to thank Prof. Nazim Madhavji and Prof. Paul Sorenson for some very fruitful discussions. We would also like to acknowledge the financial support of the Natural Science and Engineering Research Council of Canada and IRIS Network of Centres of Excellence in carrying out this research.

References

[1] V. Arora, "A granularity based architecture to represent design rationale", M.Sc Thesis, Dept. of Computational Science, University of Saskatchewan, Saskatoon, Canada, 1992 (in preparation).

[2] J.E. Greer and G. McCalla, "A computational framework for granularity and its application to educational diagnosis", *Proc. of Eleventh IJCAI*, Aug. 1989, pp 477-482.

[3] J. R. Hobbs "Granularity", *Proc. of Ninth IJCAI*, Aug. 1989, pp 432-435.

[4] K. E. Huff and V.R. Lesser, "A Plan-Based assistant that supports the software development process", *ACM SIGSOFT Soft. Eng. Notes*, 13, 5, Nov. 1988, pp 97-106.

[5] A. P. J. Jarczyk, P. Loffler and F. M. Shipman, "Design Rationale for software engineering: A Survey", *Proc. of 25th Hawaii Int. Conf. on System Sciences*, Jan. 1992, pp 577-586.

[6] B. Liskov and J. Guttag, "Abstraction and Specification in program development", MIT Press, 1986.

[7] G. McCalla, J. Greer, B. Barrie and P. Pospisil, "Granularity Hierarchies", *International Journal of Computer and Mathematics*, Vol. 23, No. 2-5, 1992, pp 363-375.

[8] M. S. Moriconi, "A Designer/Verifier's Assistant", *IEEE Trans. Soft. Eng.*, 5, 4, July 1979, pp 387-401.

[9] N. H. Madhavji, K. Toubache and W. Hong, "Towards building reliable process models", Tech Report - School of Computer Science, McGill Univ., March 1991.

[10] C. Potts, "A Generic Model for Representing Design Methods", *Proc. 11th Int. Conf. on Soft. Eng.*, IEEE Computer Society Press, May 1989, pp 217-220.

[11] C. Rich and R.C. Waters, "The Programmer's Apprentice", Addison Wesley, Reading, Ma, 1990.

[12] H. W. J. Rittel and M. M. Webber, "Dilemmas in a general theory of planning", *Policy Sciences*, 4, 1973, pp 155-169.

[13] P. G. Sorenson, J. P. Tremblay and A. J. McAllister, "The Metaview system for many specification environment", *IEEE Software*, 5, 2, Mar. 1988, pp 30-38.