THE DESIGN OF KNOWLEDGE-BASED SYSTEMS FOR MANAGING ILL-STRUCTURED SOFTWARE PROJECTS

by

Rajan Srikanth
Leonard N. Stern School of Business
Information Systems Department
New York University
90 Trinity Place
New York, NY 10006

and

Matthias Jarke
Lehrstuhl fur Informatik
Dialogorientierte Systeme
Universitat Passau
P.O. Box 2540
8390 Passau, West Germany

July 1989

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

CRIS #211
STERN #89-80

Matthias Jarke's work was supported in part by the European Commission under ESPRIT contract 892 (DAIDA), and by the Deutsche Forschungsgemeinschaft under grant no. Ja 445/1-1. The contributions of Udo Hahn, Manfred Jeusfeld and Thomas Rose to the conceptual software process model used in this paper as well as discussions with Vasant Dhar are greatfully appreciated.

Abstract

Current planning and control procedures for large-scale software projects are not sufficiently equipped to deal with changing or imprecise requirements, resource breakdowns, unexpected delays, etc. We propose a solution for managing change in projects, based on a semantic model of the software design and development processes. At the heart of this technique is the formation of *islands of project knowledge* in a way that facilitates dealing with most design and plan revisions locally. A protocol for interactive change management is presented that advocates need-based formation of coalitions between islands as a means for graceful degradation in the place of strict hierarchical control. The results of initial empirical investigations of the usability of the approach and plans for its continuing evaluation are also reported.

Keywords: project management, coordination systems, distributed knowledge-based systems, group decision support, proactive planning, software process model, software knowledge base, software environment.

1 Introduction

Every organization undertakes "one-shot" activities that are not routine or repetitive, requiring collaborative work among two or more participants to achieve a certain purpose. We commonly refer to such undertakings as **projects**. The dimension along which projects vary most is structuredness. A project is *well-structured* if its objectives or deliverables can be specified with precision and detail apriori, and a definite statement can be made about the sequence of activities, and kinds of resources needed for this purpose. Projects in manufacturing and civil engineering for instance, tend to be well-structured since they have clearly specified goals laid out in blueprints, and well-defined means to deliver these objectives. In this paper, we are interested in projects which are less structured. We are concerned in particular with situations where a strict, contract-like specification of the project cannot be made in advance, but evolves as the project is undertaken.

Less structured projects such as the development of large-scale software systems differ fundamentally from other projects because they are much more information-intensive. Since they are ill-defined, a variety of assumptions must be made even at the very beginning to impose a semblance of structure on the project. Events that occur during project implementation and interactions with a changing external environment, periodically force reevaluation of these assumptions and may result in a revision of project objectives and strategies. The complexity of managing such poorly defined work enterprises is further worsened by the need for exchange of information between multiple, semi-independent participants who exercise control over different parts of the project. Considering the predominant role of information - how it is continuously generated, processed, and assimilated - it is apparent that the management of ill-structured projects requires techniques to capture this information, and support its characteristic patterns of usage and modification.

Current project management systems were primarily designed for use in well-structured projects and therefore assume that there is only a limited need for ongoing information management. The network of project activities and requirements of resources are treated as given and not subject to change. Activity durations and resource availabilities may however vary as a result of environmental or chance factors. As a result, information management is limited to techniques for collecting, assimilating, and propagating information about delays and resource breakdowns.

The advantage of using such a project model is its computational simplicity. Powerful mathematical programming algorithms have been developed for finding optimal schedules and resource allocations for a given project network [BMR89]. Ill-structured projects however, function open-loop and evolve constantly. Since every aspect of these projects is subject to change in an unpredictable way, efficiency is less of a concern than effective response to change. The simple representation used by classical approaches does not provide adequate information for project control decisions. Knowledge of the 'logic' behind project designs and plans, and mechanisms for reviewing them in the light of new information are necessary, but not available.

This research advocates a new approach for the management of unstructured projects in general, and large-scale software development projects in particular. It is anchored on strategies for the representation and management of project-related information that draw upon observations from an empirical study that we conducted, and work that the we have undertaken in developing knowledge-based approaches for software development [DAIDA88] and management of change in projects [SRI89]. We advocate a two-pronged strategy:

1. Support for the "technical" aspects of change (revision of design specifications, etc.). This could take the form of creating and managing a knowledge base of project information, that captures the 'logic' or rationale behind design and plan decisions. A software process and project model for structuring such a knowledge base developed in [JJR89] is used in this paper. And,

2. Support for the "managerial" aspects of change (minimizing the impact of unanticipated developments, building flexible plans & designs etc.). This may be done by forming **"islands"** of project knowledge and using these as a basis for distributing project responsibility. A methodology for partitioning the knowl-

edge base along these lines, is also offered in this paper.

Each *island* is an almost independent information-handling and control unit - a sort of 'organization-in-the-small' - a set of project activities linked to other like units by virtue of commitments requested or made. These commitments typically concern design specifications or plan deliverables. The *islands* are formed in a way that most knowledge relevant to the design and planning of a given part of the project, is available locally within the *island.*

Formulation of the project control problem as an attempt to localize change within *islands*, allows us to apply techniques that we develop, recursively to higher level aggregates - collections of *islands*, multiple interdependent projects, etc. We turn now to a brief review of other research that has sought to introduce knowledge-based technologies into project management over the past few years.

One category of approaches deals with the development of conceptual models for capturing project-related information. Most of these approaches are grounded in the traditional planning-for-purpose-of-scheduling framework, that views projects in terms of activities and precedence relationships. Bimson and Burris [BB88] for instance, provide a frame-based reformulation of classical project modelling techniques along with some limited document management capabilities (cf. also [HOF88]). In a similar but more extensive fashion, Callisto [SFG85] provides formal representations for notions of project state, activities, and goals. Abstraction mechanisms allow the description of projects in these terms at varying levels of detail. Even greater detail is captured in the PIMS model [VAU88]. It is interesting to note that all of these proposals use essentially the same interval-based model of time proposed by [ALL83]. Extensions of the interval-based model for specific use in software project management have been studied by Ladkin and others at the Kestrel Institute [GJLP87].

A second class of approaches focus on appropriate *architectures* for developing knowledge-based systems to manage software projects in specific domains. Kurbel and Pietsch [KP88] propose a hierarchical architecture for the management of evolutionary software projects (such as expert systems development). There are three interacting levels in this hierarchy: strategy, project structuring (work breakdown and communication), and project operation (time management). Feedback from lower levels to higher levels is posited, but no formal model of this interaction is offered. Kerzierski [KED84] addresses the issue of knowledge base design. He proposes integrating knowledge of project structure with knowledge of the target system to be developed. Subsequent work within the KBSA project [JDL88] extends this project knowledge base by adding application-specific software engineering heuristics and procedures, aimed at automating project planning.

A final class of approaches that apply knowledge-based technologies to some aspect of project management go by the name of collaborative-work tools. They emphasize ongoing control of distributed work in offices or projects, rather than planning activities per se [FIKE82]. Their objectives is to structure the communication undertaken between participants in collaborative work, by providing a set of primitive message *types* embedded in enhanced electronic mail systems (the COORDINATOR [WF86]). Message types specific to software engineering such as bug reports, have also been developed [DDSVZ86, KED84]. The LEONARDO project at MCC [BCE-GRS86] applies this approach to the design of a face-to-face multimedia meeting support environment for software projects, whereas the CoNeX [HJ89] and CoAUTHOR [HJKFP89] projects at Passau University aim at the conceptual integration of software project management, software development environment, and software documentation development and maintenance in a distributed real-time multimedia conferencing setting.

In summary therefore, most existing work in developing knowledge-based systems for project management uses the traditional scheduling perspective as the point of departure. While allowing a more detailed representation of activities, resources, milestones, and other domain-specific project knowledge, the proposals just reviewed do not quite address the needs of ill-structured projects. They do not capture the dependencies between objectives, activities, and resources, that underly the logic of project design and plan decisions. As a result, support for change management is limited. Collaborative-work tools on the other hand, provide a basis for structuring the communication undertaken to manage change, but do not offer any guidance for design or plan revision decisions. The ideas presented in this paper differ from both of these in the following respects:

- We have developed a methodology for change management that integrates traditional scheduling concerns with the need for control and coordination in the face of change.
- We extend the basic conceptual model of projects used by techniques such as CPM and PERT, to capture the logic of project plans in order to facilitate pro-active project planning. These extensions are 'consistent' with the existing model and can therefore be grafted on to currently available project management systems. And
- Finally, the project model and change management methodology offered in this paper have strong empirical roots, and are being subjected to careful validation. The completeness and understandability of the model, and the effectiveness of change management with this methodology, have or are being empirically evaluated.

The remainder of this paper is organized as follows. Section 2 outlines the empirical background, and the basic principles we propose for the design of knowledge-based systems for managing ill-structured projects. In section 3, a formalization of these principles in the context of software projects, is sketched out in terms of a knowledge representation language based on semantic networks. Section 4 describes a project management methodology based on this formalism, its implementation through a knowledge-based support system, and a strategy for empirical evaluation. Section 5 summarizes conclusions and directions for future work.

2 Requirements for a Project Knowledge Base

We argue that ill-structured projects are information intensive, and that effective management of information is critical for their success. Information management requires the development of techniques for: (1) capturing all appropriate information,

(2) representing it in a form that facilitates effective usage, (3) processing and utilizing the information to maximum benefit, and (4) ensuring the integrity of the 'state of information (or knowledge)' over time. In this research, we develop an approach for the management of ill-structured projects by addressing these requirements.

The first question that must be addressed is "what information must be captured?". A starting point for finding an answer to this question is to examine what we mean by a project, more closely. This research takes the position that a project is a purposeful, non-routine activity that involves collaboration among people. The process of software development is clearly a point in case. By defining projects in this manner, we make certain implicit assumptions about the nature of the project world - the entities of interest, and dependencies between them. In this section, we show how conventional views of projects differ from this perspective and are inadequate for our purposes. We offer a formal model or *ontology*, of the project domain and empirically demonstrate its adequacy for capturing information used in project planning. We then develop prescriptions for how this information must be represented and organized in a project knowledge base, for the effective management of change. One important characteristic of these prescriptions is that they facilitate knowledge base maintenance and ensure greater integrity of stored knowledge, over time.

2.1 Conceptualizations of Project Management

Research that has addressed project management concerns to date, belongs to one of two classes: (1) network planning techniques, and (2) collaborative-work research. A brief overview of these areas is offered below.

Network planning techniques (CPM/PERT) are typical of the *classical* approach to project management. They model a project as a given network of activities linked by precedence relationships. Resources are seen as constraints on finding best schedules, and allocation of resources is undertaken using mathematical programming formulations, to maximize utilization or minimize project duration *(Figure 1)*.

Though the classical approaches provide powerful algorithms for efficient scheduling, the information they capture is limited to activity precedences, duration and

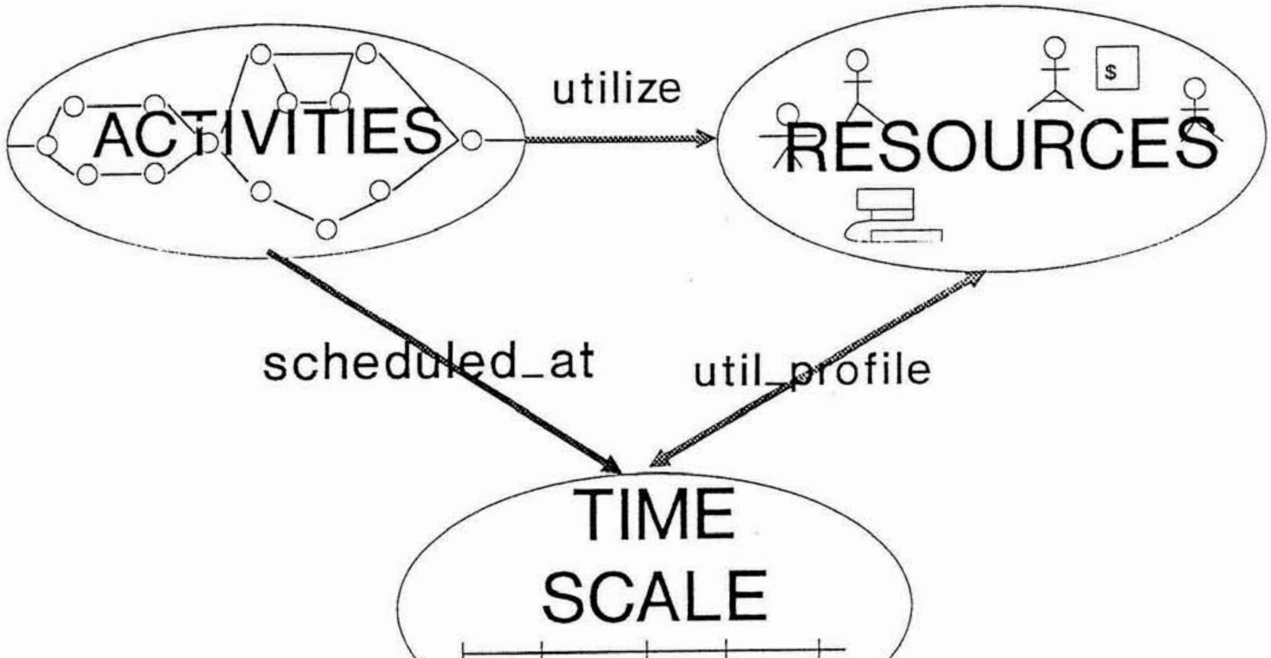

Figure 1: Classical view of Project Management

resource estimates. This surface representation of available project knowledge is probably appropriate for well-defined and repetitive projects such as in manufacturing or engineering. However, it proves inadequate for Information Age projects like large-scale software development where there is a need to manage a wide variety of information. For instance, the goals of a project are typically not clearly specified and evolve during implementation. Project plans made in these situations reflect assumptions that have been made to impose some structure. As goals evolve, the assumptions must be reevaluated and plans revised accordingly. Yet another source of difference is that people resources utilized by these projects tend to fall under the growing category of "knowledge workers", as different from labor or clerical personnel. They collaborate with each other, exchange information, and exercise control over different parts of the project. Resources such as these cannot be treated as passive scheduling constraints.

Techniques for modeling collaboration between people, and computer-based tools for supporting collaborative work have become an active area for research in the recent past [FIK82] [WF86]. Collaboration is viewed as a process in which individual participants initiate requests, and make *commitments* to fellow workers concerning the performance of activities at specified points in time *(Figure 2)*.

Sophisticated tools have been developed for structuring the communication processes involved in collaboration [COORD86]. Though such tools offer added value by providing formal techniques for electronic messaging between participants, there is little support for the complex decision-making undertaken to plan and control collaborative work.

Classical approaches to project management, and techniques for modeling workgroup collaboration therefore represent different extremes on the spectrum of project ontologies. It appears however, that neither of these conceptualizations succeeds in capturing the wealth of knowledge typically used in the planning and control of Information Age projects.

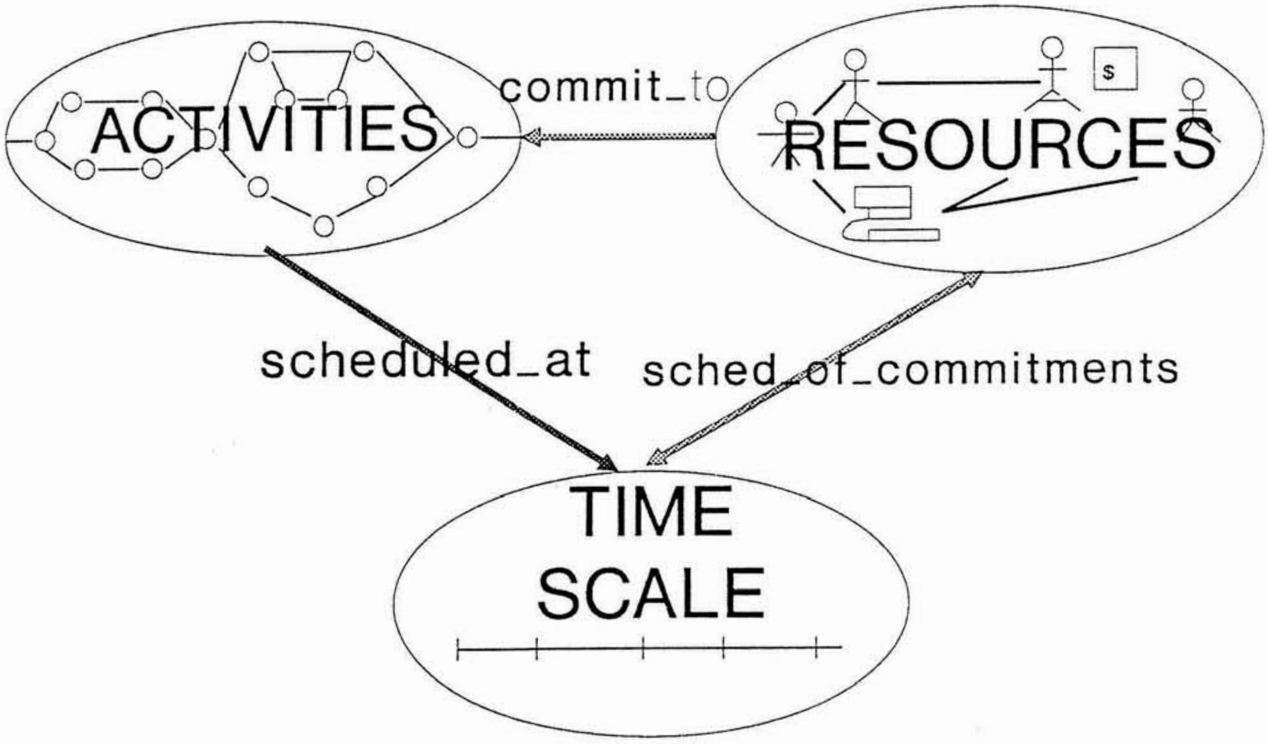

Figure 2: Collaborative Work perspective on project management

2.2 Some Empirical Evidence

We conducted a set of empirical studies to better understand the knowledge used by project managers planning software projects.

Subjects: Ten part-time graduate students (6 female, 4 male) of an advanced capstone Information Systems elective participated as subjects in this study. These subjects had an average 2.7 years of experience in managing systems development projects at work. Their business backgrounds varied but a majority worked in the financial services industry.

Task: A case study that described problems faced by a company which manually stored and retrieved personnel document microfiches was presented, and an outline for a computer-based alternative was offered. Subjects were then asked to imagine they were in-charge of the systems group, and required to prepare a plan and schedule for designing, developing and implementing a computerized document maintenance system.

Data Collection and Analysis: Collection and analysis of think-aloud *protocols* [ES84] is a technique that has gained wide-spread acceptance in experimental psychology as a method for studying cognitive information processing. In this study, subjects were instructed to "think aloud" while planning the project, and to voice all assumptions and thoughts. These think-aloud protocols were recorded on audio tape and subsequently transcribed. An analysis of the transcripts was performed to identify the different kinds of knowledge used, and to obtain insights into the process of project planning.

Findings: Analysis of the transcripts clearly establishes that subjects engage in more than just specifying activities and precedences. For instance, subjects made use of a variety of information to reason about project objectives/deliverables and activities:

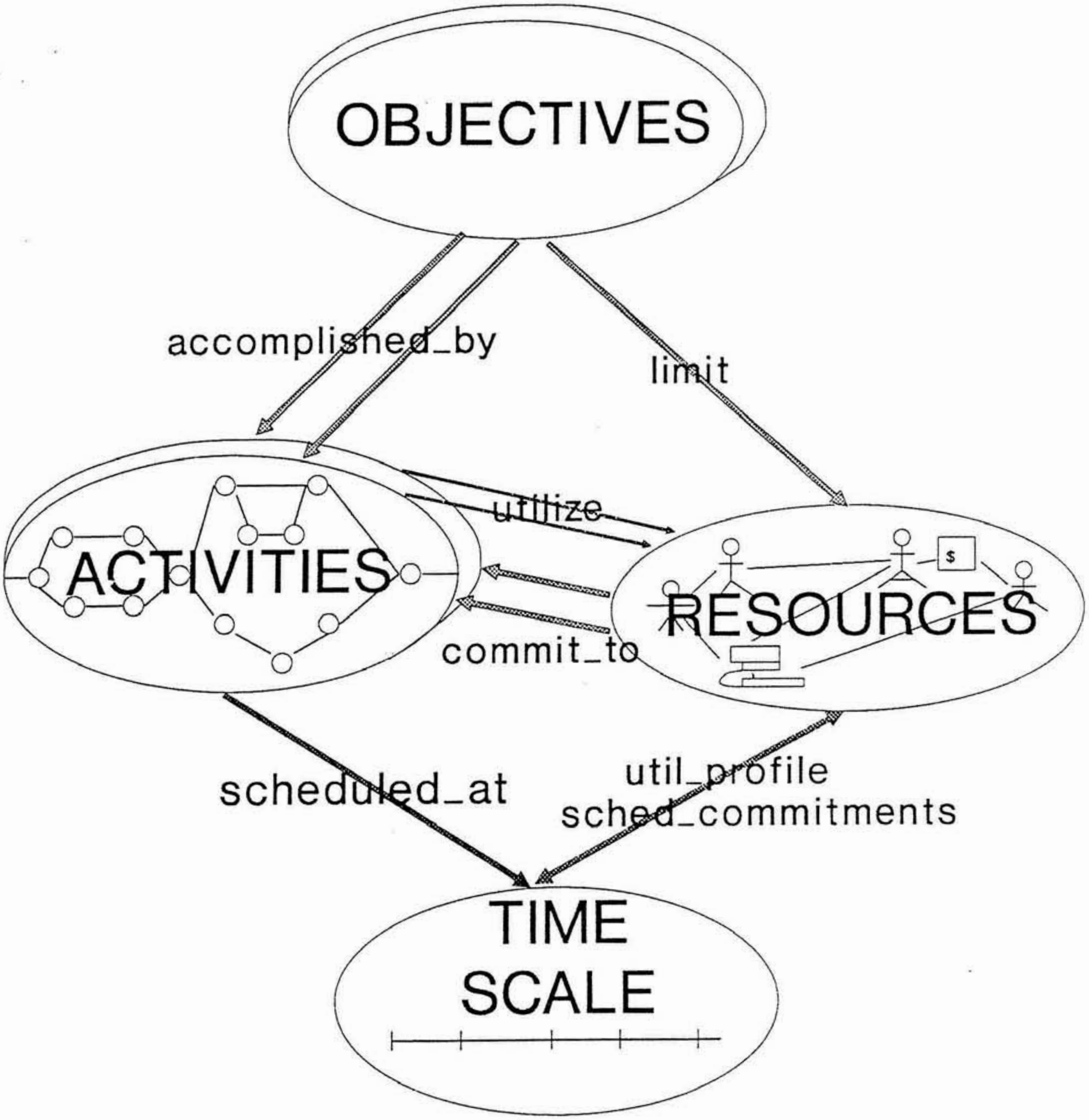

Figure 3: Conceptual Model for Information Age projects

> *"You have to make a decision whether to use this ADF facility or the IMS. ADF seems to be tailored to the application (from the description). If you want to go to ADF, you will have to train your people."*
>
> *"You might also want to do a demo to your users - I put as a task 'Convincing users' (to go with the system), because the users have a different idea in mind (according to the case)."*
>
> *"After we determine what the index is, then we also need to determine the conversion strategy. We have a system in place today and we need to convert it to a new system. We have to determine how we are going to do that."*

In addition, subjects considered several kinds of relationships between activities, and made assumptions or statements of preference while determining precedence relationships.

> *"Determining the requirements - we can determine the requirements before we learn the system, but it would'nt be a good idea. We should write requirements understanding the tool we have to work with.."*
>
> *"Designing the screens, I think, does not rely on the design of the index because the internal database could be structured in a certain way and the screens could be structured in a different way.. I am going to put that parallel to Designing the index, but I am going to put it after Convincing the users because we want the user's to help us design screens.."*
>
> *"..Conversion - if we are using a PC, conversion can start as soon as we have designed the index, and it does not depend on the screens and the reports."*

The implications of these findings for the development of techniques for managing software projects are the following: (1) a variety of knowledge other than activities, and precedence relationships underly the reasoning processes involved in the development of a project plan, (2) this knowledge may be systematically captured, and (3) availability of this knowledge may be used to pro-actively plan projects to accomodate change, and subsequently to generate strategies for control.

2.3 Enhanced model of project knowledge

Based on our empirical observations, we developed a model for conceptualizing Information Age projects *(Figure 3)*. This view of the project world is adapted from the ontological model of projects proposed in [SRI89].

Descriptive knowledge about any domain including projects, may be captured by modeling entities of interest, and the dependencies that exist between these entities. Project objectives or deliverables, the activities that accomplish these objectives, the resources that are used for this purpose, and the time scale or time horizon over which the project is scheduled, are the types of project *entities* that we are concerned with.

Two types of decisions are made during project planning: design decisions, and plan decisions.

- **Design decisions** concern the determination of project objectives or specifications (*design objects*), and the activities and resources needed to "deliver" these objectives (*design activities*).
- **Plan decisions** involve the determination of precedences between activities, and resource allocations, based on assumptions about work dependencies or relationships between activities.

Documentation of the knowledge underlying design and plan decisions, is critical for managing change in poorly defined, evolving project scenarios. An example is the case of developing large software systems.

To document this knowledge, it is also necessary to define the different kinds of dependencies that may exist between project entities. There are primarily two categories of dependencies: *associations* representing dependencies between entities of different types, and *relationships* or dependencies between entities of the same type. Associations of particular interest are the ones between objectives and the activities they are accomplished-by, associations between resources and the activities they commit-to, etc.

Relationships can also be of several types. Srikanth [SRI89] took a work-centered view of projects, and based on observations from the empirical study, identified the following 4 kinds of *activity*-relationships: (1) output-input relationships representing one activity's dependence on another for needed input, (2) facilitatory relationships where the performance of one activity presumably makes another easier, (3) coordination-reqd relationships where reciprocal influences between activities makes a temporal overlap necessary, and (4) shared-resource relationships between activities that use the same resource in adjacent time intervals. This categorization of activity-relationships refines and extends the classification of work dependencies proposed in the literature on Organizational Design by Thompson [THO67].

This characterization of project knowledge was evaluated through a follow-up study. Six experienced project managers - three Civil Engineers and three Systems Development professionals (with > 5 years experience) - were asked to think out aloud while they planned two different test projects: a kitchen renovation, and the development of a computerized document maintenance system. The transcripts of the think-aloud protocols were studied by a panel of 3 independent judges, who coded the information heeded by subjects, in terms of these relationships. Inter-rater reliability, used as a measure of how appropriate the relationship categories were, turned out reasonably high. The judges also agreed that in their subjective evaluation, the model seemed complete and adequate for representing information used in generating project plans.

The above taxonomy of relationships focusses on knowledge underlying *plan* decisions. In ill-structured projects, we have seen that a second class of class decisions - project *design* decisions - are also of interest. This paper extends the taxonomy by incorporating dependencies that play a role in decisions about design: (1) When a set of activities is specified as necessary for "delivering" a single design object or objective, each member of the set is conjointly dependent on the others. And, (2) two activities are logically dependent when the method or way of doing one activity determines the choice of method for the other.

When project knowledge is represented in terms of this model, we succeed in capturing the 'whole picture'. It facilitates documentation of the logic behind project decisions. For projects that evolve over time, this kind of documentation allows managers to consider alternative responses to change in the light of earlier decisions about the project design and plan.

In terms of the information management problem posed earlier, this observation gives us a basis for an answer to the second question: "in what form must information be represented to facilitate its effective usage?" We argue that knowledge of the logic underlying project decisions must be represented in a knowledge base so that it it is available for consultation prior to making revisions in design and plan.

2.4 Design goals for project knowledge bases

The first and primary goal in designing a knowledge base of project information, must therefore be to capture the different kinds of knowledge embodied in the enhanced project model. A discussion of how some of this information can be effectively utilized to generate strategies for change management can be found in [SRI89].

That however, is not all. In addition to the static characteristics of the knowledge base, it is also essential to consider its dynamic traits: patterns of usage and modification, maitaining integrity of knowledge over time, etc. Ill-structured projects evolve and are prone to one or both of the following kinds of changes:

1. Changes in design specifications (changes in objectives or available design alternatives). And,

2. Changes in plan specifications (changing activity time estimates and resource availabilities).

Since these projects are inherently poorly defined, changes in objectives or scope, and deliverables, are unavoidable. When objectives or deliverables change, design decisions must often be revised, potentially leading to chain-reaction revisions to logically or conjointly dependent design activities.

Likewise, when unanticipated external events such as resource breakdowns and overruns cause delays, schedules are revised and resources reallocated. The web of work dependencies such as output-input relationships, shared-resource relationships etc., propogate these changes, leading to system-wide revisions.

Such wide-spread revision of design and plan decisions is typically unacceptable from an operational standpoint since it makes coordination within and across interrelated projects very difficult. It also gives rise to the need for more frequent and widespread "repair" of the knowledge base, to maintain the integrity of information stored.

Designers must therefore consider the following while developing knowledge bases for supporting ill-structured, Information Age projects: (a) changes in design and plan are inevitable, (b) knowledge of design and plan logic facilitates response to change, and (c) project decisions about design and plan, are interdependent in a variety of ways - the revision of one could possibly cause chain-reaction changes in others. The question that must now be addressed is: *"how must we design knowledge bases to support management of projects such as these?"*

Computer-based systems that support organizational information processing are either designed to be compatible with existing organizational practises, or with a view to modifying them as seen fit. Here, we are interested in the design of knowledge-based systems for management of uncertainty and change in project organizations whose work is poorly defined.

From an organizational view-point, it is clearly desirable that such projects be managed "pro-actively", by increasing the ability to respond to unanticipated events [ACK81]. It is also essential to find ways to limit chain-reactions, or localize the effects of change in design or plan decisions. Systems theory [SIM62], and the literature on Organizational Design [GAL73] advocate the creation of self-contained units to reduce the need for wide-spread exchange and processing of information in such circumstances. A knowledge-based system that aims to assist ill-structured project organizations may be designed in a like spirit.

We argue that chain-reaction revisions of project knowledge bases can be pro-actively reduced by partitioning the knowledge base into minimally "related" groups of design and plan decisions. If an effort is made to confine the effects of changes in design and plan to be within one or more of these **"islands"** of project knowledge, the extent of knowledge base maintenance required can be significantly reduced.

This knowledge partitioning also assists in the organizational aspects of change management. When the project knowledge base is designed and constructed in this manner, knowledge that is required for control is distributed. Responsibility and ownership of each knowledge base module may now be assigned to different knowledge workers in the project organization, who will then have complete authority for local project control decisions. Formation of these almost-independent groups of design and plan decisions, therefore defines **islands of control** for managing a project (see [SRI89] for a more thorough discussion of the benefits of forming *islands of control* to manage change).

The second goal in the design of systems for supporting poorly defined, evolving projects, must therefore be the partitioning of the project knowledge base into almost-independent segments or modules.

In summary therefore, we argue that knowledge-based systems for managing ill-structured projects may be designed as follows:

1. Step 1: Making use of the enhanced project model as a framework for capturing and representing information about project design and plan decisions, in a knowledge base.
2. Step 2: Utilizing this knowledge, and a definition of "relatedness" to identify minimally related groups of design and plan decisions. And,
3. Step 3 Modularizing the knowledge base into **islands of project knowledge** along these lines.

The next section sketches a semantic network formalization of such a model as a basis for an implementation effort.

3 Software Project Knowledge Representation

The formal software project model sketched below is intended to represent content, process, and project control for design tasks in a uniform framework suitable for co-operative work. Although any programming language could be used to *implement* such an approach, a knowledge representation language with powerful abstraction mechanisms specifically tailored for requirements modelling allows for a concise initial *specification.* In this paper, we use a subset of the knowledge representation language CML/Telos for this purpose. Actually, a related model which emphasizes different aspects of project management (e.g., negotiation support [HAHN89]) is being implemented in the CoNeX system on top of the ConceptBase knowledge base management environment [JJR88] that directly implements CML.

3.1 Basic Concepts

CML/Telos is a hybrid knowledge representation language that integrates predicative rules and constraints and an interval-based time calculus into an object language based on highly structured semantic networks. The language has been used in various applications mainly in software requirements modelling and software information management. A full description including a formal semantics can be found in [KMSB89].

The CML *object language* fully supports the abstraction principles of classification, generalization, and aggregation. A number of built-in axioms enforce the semantics of these structuring principles. In terms of surface syntax, a knowledge base can be equivalently described as a semantic network, a set of frames, or as a combination of both. In the ConceptBase implementation, this has been exploited to offer the user hypertext-style interaction with the system [JJR88].

Classification is used to make the language extensible in the sense that it is easy to define sub-languages for specific application areas. For example, the software process meta-model and the software project meta-model below define general languages in which specific software development environments can be defined by instantiation of these meta-models. Specific software development projects, in turn, are represented as instantiations of these software development environments (see section 4 for examples). One more instantiation step allows the documentation of prototyping examples for these specific software development projects. Note that this multi-level metaclass facility goes beyond that of most other object-oriented languages. Additionally, CML's classification axioms enforce organizational principles such as referential integrity in the knowledge base.

As in other object-oriented languages, *generalization* is used to facilitate reusability and localize the effect of change by inheritance mechanisms. CML supports multiple inheritance to avoid representation anomalies.

Aggregation is intended to relate in-the-large and in-the-small development by an attribute mechanism that connects objects to each other. A unique feature of CML is that attributes (i.e., links among objects) are themselves full-fledged objects with classification, generalization, and inheritance. In the software process model, below, this feature is used to document dependencies among attributes created by design decisions.

Modular aggregation is a special kind of aggregation in which access to part objects can only be achieved through explicit import and export, as in programming languages such as MODULA or Ada or in the 'worlds' approach of [WA88]. This extension is currently being added to CML in order to support the information hiding requirements of distributed project control, as required by our approach. It also serves as the basis for conceptual version and configuration management [JJR89] required for the controlled integration of separately developed software components.

The CML *assertion language* introduces a special kind of objects into the semantic network. These objects are represented externally by strings that express first-order predicate calculus expressions. As in deductive databases, these expressions can be used either as deduction rules to infer implicit information from stored data, or as integrity constraints to enforce semantic theories not directly expressible by the structural axioms of the object language. Which of the two meanings of an assertion object is intended, is expressed by the class of the attribute link pointing to the assertion object.

The *temporal* sublanguage is also fully integrated into the data model; this, of course, substantially facilitates project modelling in which the duration and temporal relationship among (activity) objects plays a central role. Since *every* CML object has built-in time, there is no need to introduce explicit time objects as shown in figures 1 through 3. Temporal constraints can be expressed using Allen's [ALL83] interval calculus. This calculus, however, offers only simple temporal relationships such as "before", "overlaps", etc. Using the CML object language, content-oriented specializations such as discussed in section 2.3 (cause, facilitate, etc.) can be introduced. A detailed discussion of these options would go beyond the scope of this paper (see [BMR89], [SRI89], and other works cited in the introduction).

3.2 Software Process Model

As an example of the above CML facilities, we now review a basic model of software development processes fully elaborated in [JJR89]. The model, summarized in the semantic network of *figure 4a*, is actually a meta-model of software development environments that consists of three kinds of objects:

- *Design objects* represent any kind of intermediate result achieved in the software process. For example, in the DAIDA software development environment [DAIDA88], this includes requirements models (i.e., functional and non-functional system requirements as well as a conceptual model of the system environment), formal specifications, conceptual designs, the actual software, and any accompanying documentation.

- *Design decisions* characterize the planned or actual activities required during the software development process. In DAIDA, we distinguish refinement of the design objects within a particular level of representation, mapping between different levels (e.g., from design to implementation), retract decisions that correct previous refinement or mapping decisions, and release decisions that make a particular design object available to a larger group. Documenting these decisions explicitly as knowledge base objects defines dependencies to be used in subsequent maintenance decisions.

- The execution of design decisions is supported by *design tools*. Design tools are usually reusable software components specifically designed to support a particular class of design decisions. In the DAIDA environment, tools are triggered by the user by suitable attachment to particular design objects [JJR89].

Since the software process data model is represented in CML, the full power of object language, assertion language, and time sublanguage is available to the representation of actual software development environments (instances of the meta-model) and software development projects (instances of those software development environments). For example, the time calculus can be used to represent versions of certain software components, and the assertion language for expressing constraints such as those generated by certain commitments in the process model sketched in section 3 of this paper. Most importantly, modular aggregation can be used to isolate certain configurations of design decisions and their intermediate results so that only their external commitments remain visible.

3.3 Software Project Model

The software process data model is intended (and used) as a means to represent a software development environment from the perspective of the *contents* of design tasks. Surprisingly, much of the model remains valid when viewed from a project management perspective. However, knowledge base objects take on additional interpretations:

- Design objects can be viewed as *goals* or *deliverables* whose temporal validity characterizes the (planned or actual) status of a project. (Besides, a particular new kind of design object, the project plan itself, should be introduced, together with design decisions that create or change the plan.)
- Design decisions represent the *project activities*; their conceptual representation is reduced from a rich, content-oriented model to information about their temporal duration, temporal interrelationsships, and tool requirements. Even qualitative information about the design decisions may be rather different, when viewed from the project management perspective, as indicated by Srikanth's taxonomy of design dependencies; possibly, deduction rules can be used to infer these relationships from the content description of design decisions.

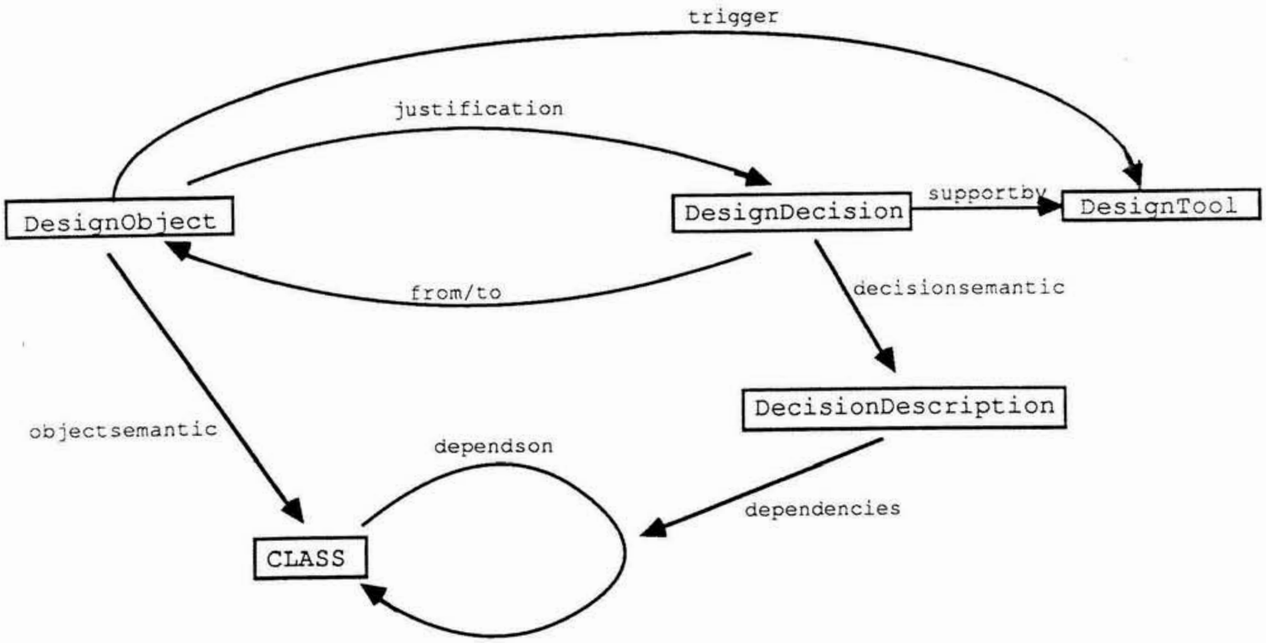

Fig. 4a: Software Process Model [JJR89]

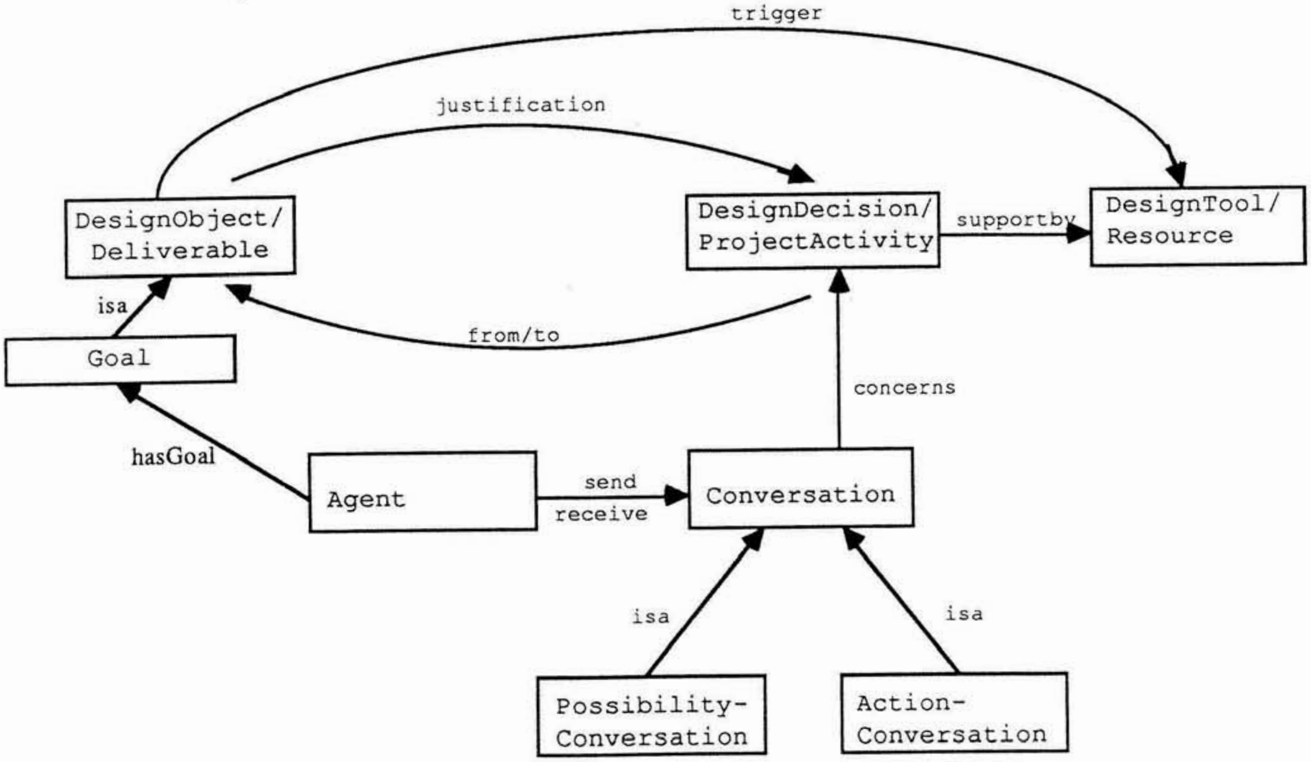

Fig. 4b: Software Project Model

- While the process model records tools for eventual explanation and reuse in the maintenance phase, the project model views them as (scarce) *resources* that have to be scheduled and may cause costs. This may require the recording of tools/resources that are of no interest to the software process model (e.g., money).

In summary, although the basic software process meta-model remains useful as a structuring mechanism for software project knowledge as well, different viewpoints [AS84] have to be offered when we intend to integrate software development support from the engineering and from the project management perspective. In CML, this can be achieved by the use of deduction rules.

Additionally, however, decentralized project management requires modelling of the *agents* that control each island of project knowledge, and of the *commitment* structures between these agents, as defined by the project breakdown into *islands*. Like all other CML objects, agents and commitments can be organized in modular aggregates of related objects. In the case of agents, this allows the modelling of project group structures; in the case of commitments, it defines how individual messages can be composed to entire conversations. Analogously to speech act theory [DDSVZ86, WF86], we distinguish as specializations (using the generalization abstraction of CML!) conversations for possibilities in the project planning phase, and strictly controlled conversations for action in the project execution phase. Details of a negotiation model for this kind of networking are explored in the CoNeX system [HJ89]. For the purposes of the distributed project management approach discussed in the remainder of this paper, the level of detail given in *figure 4b* should suffice.

Summarizing the results of this section, we can see that a relatively simple meta-model of software processes and software projects can already represent a large variety of aspects in software engineering and software project management. In a practical system, however, the user may easily get lost in the intricate details of such a knowledge base. We therefore emphasize the need for powerful viewpoint facilities. In particular, the system should offer specialized browsing, zooming, and editing facilities for the viewpoints of

- any agent (or group of agents) controlling an island of project knowledge
- content-oriented software development activity
- resource-oriented project control activity

as well as many others, with changes being propagated from one viewpoint to related ones. Prior to the availability of such a generalized software environment (aspired but hardly achieved by numerous so-called IPSE – Integrated Project Support Environment – projects), a much simpler implementation appears advisable. Such an implementation, together with an associated project management methodology, is discussed in the next section.

4 Pro-active Management of Software Projects

Large software development projects are notoriously difficult to manage, and are routinely plagued by schedule and budget overruns [BRO82]. In this section, we outline a methodology for pro-active planning that operationalizes the *islands of control* concept in the context of the software project model. We then present in protocol form, the strategies that could be used as part of this methodology to interactively manage change in software projects. Finally, we discuss an implementation of this methodology that is currently underway. The presentation will be largely by example; a more formal treatment is given in [SRI89].

4.1 Overview of the Approach

In Section 2.3, we presented an argument for the modularization of project knowledge bases as a means for localizing the effects of changing design and plan decisions. The objective of such a modularization was to facilitate partitioning of the knowledge base, and limit the extent of design and plan maintenance occassioned by revisions.

Synthesis of such modules or *islands of project knowledge*, involves grouping "related" design and plan decisions together, and requires knowledge of the semantics of the software project model. If the modules are formed in a way that minimizes interdependencies or "coupling" between modules, there would be a greater probability of change being contained within a module. The concept of forming minimally coupled modules to facilitate change has strong roots in disciplines as varied as Organizational Design [GAL73], and Structured Systems Development methodologies [YC78].

In the spirit of the **islands of control** approach, when unanticipated external events cause revisions in design or plan decisions, the first response is to attempt to limit revisions locally within affected modules. If this fails, a 'coalition' of affected modules is formed and this is treated as a new, larger *island of project knowledge* within which change may be contained.

In some situations, changes and revisions are so far reaching that localized control is inadequate and there is a need for overall project reorganization. The knowledge base assists in this reorganization by providing documentation of prior design and plan decisions. The content and structure of the knowledge base itself must however, also undergo revision when the project is re-designed or re-planned.

The rest of this section deals with how we operationalize these ideas. First, consider the following project scenario which we will use to illustrate the discussion: ABC Inc. would like to computerize their personnel document maintenance function, currently being done manually. They would like to have a "user-friendly" system for storing employee data, that also provides facilities for information retrieval and generation of reports. For the system to be operational, it would also be necessary to convert all existing documents to the computerized format. A preliminary specification of the design would perhaps look like *Figure 5a.* The leaf nodes of this *design hierarchy* represent the specific design objects, or deliverables of interest. *Figure 5b* shows the design decisions of ABC's systems development team: the activities that they believe must be undertaken, in order to accomplish or "deliver" the different design objects. Several assumptions about design dependencies (*Figure 6*) underly these decisions. Plan decisions such as a project schedules and resource allocations, are also made likewise after considering work dependencies (*Figure 7*).

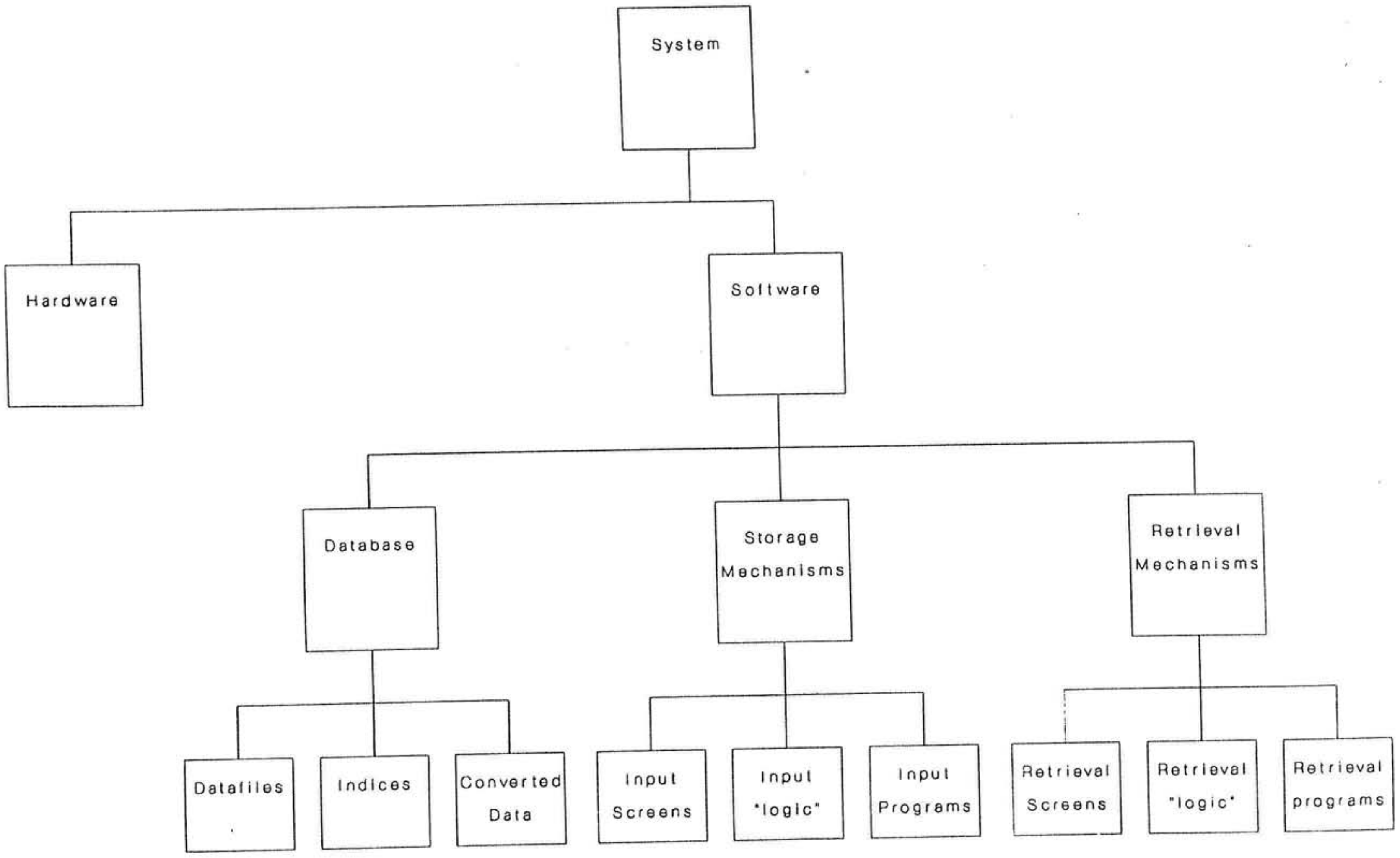

Figure ~~4~~5: Design Hierarchy

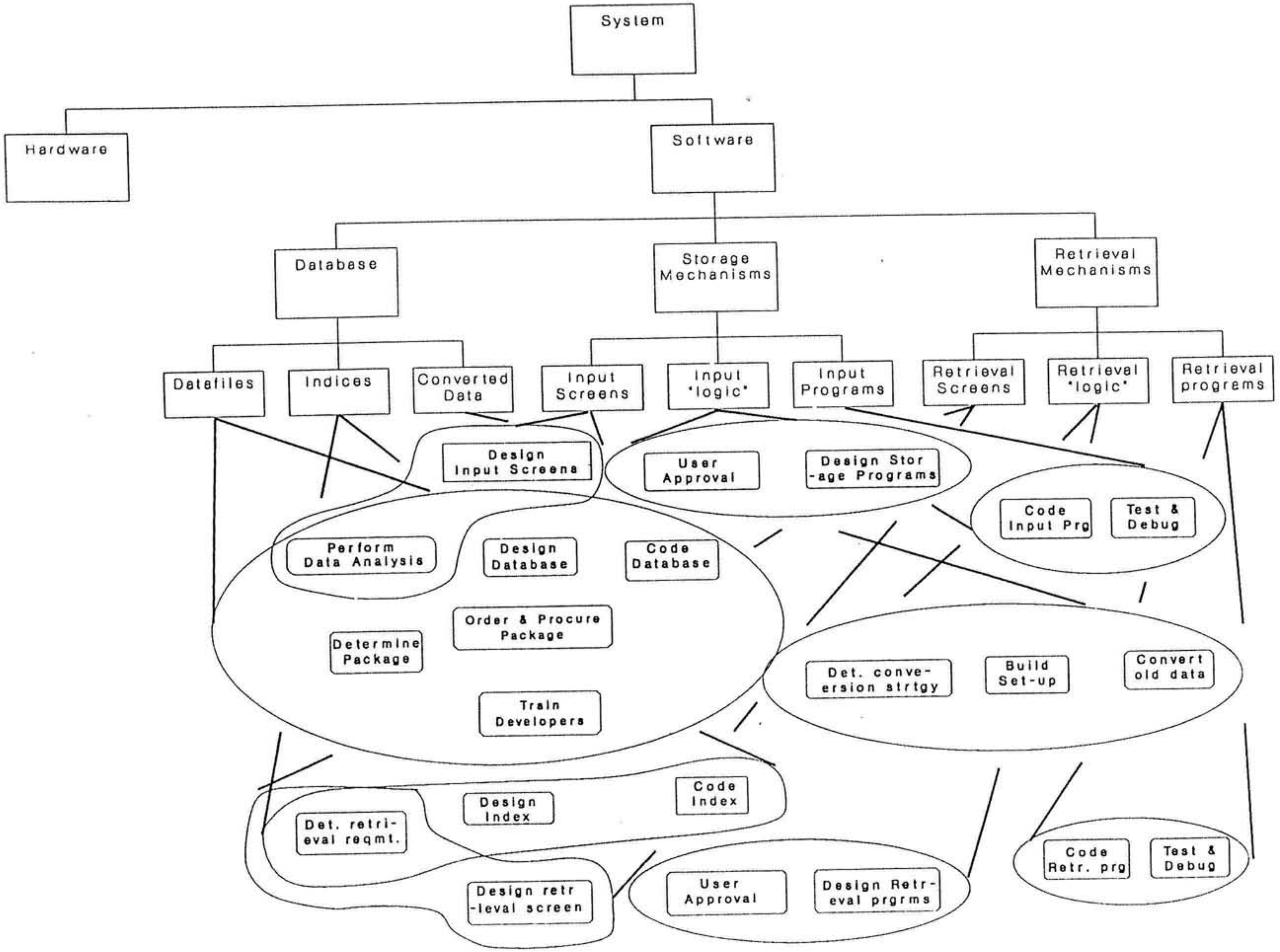

Figure 5b: Design Activities

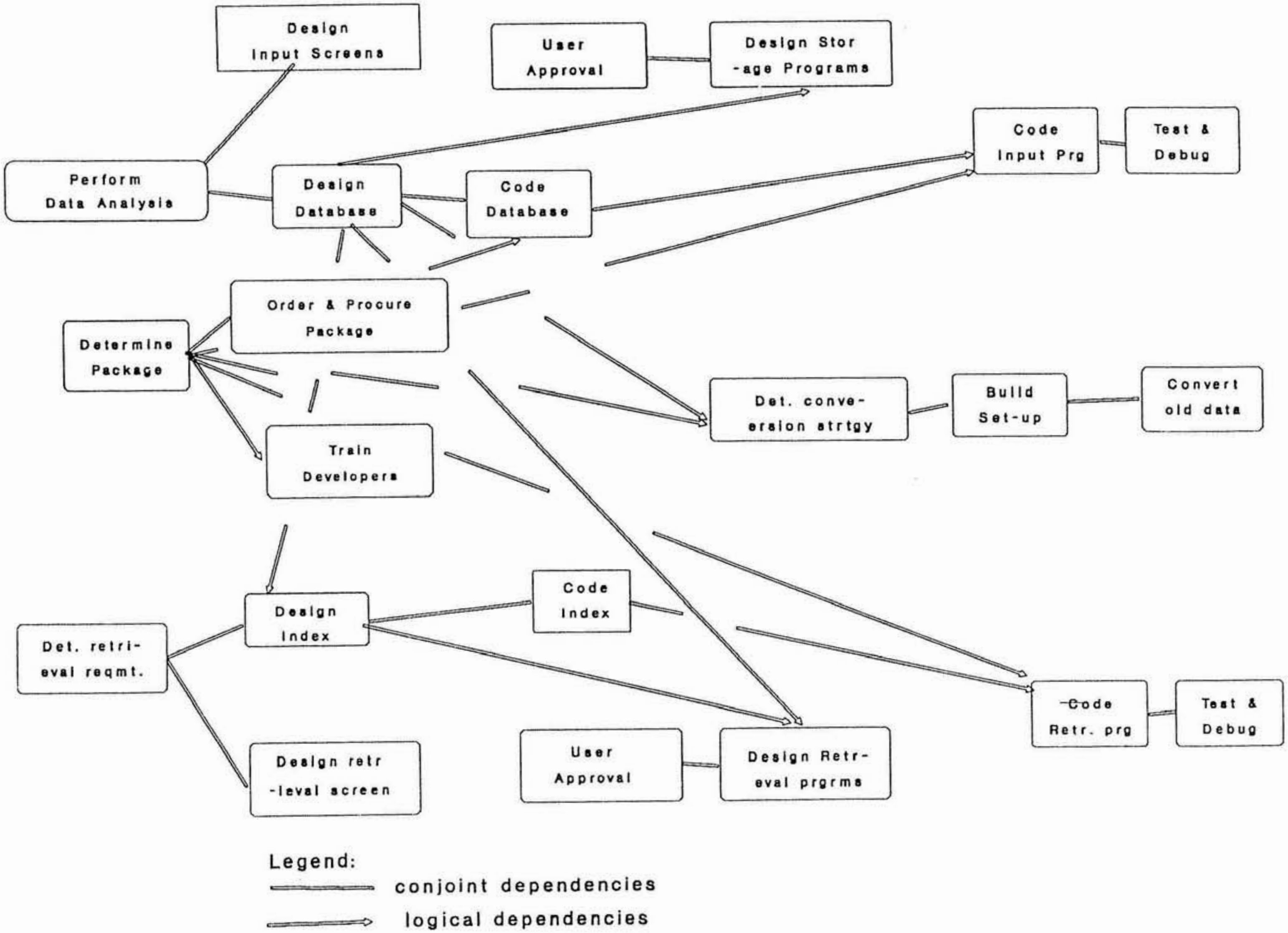

Figure 6: Design dependencies

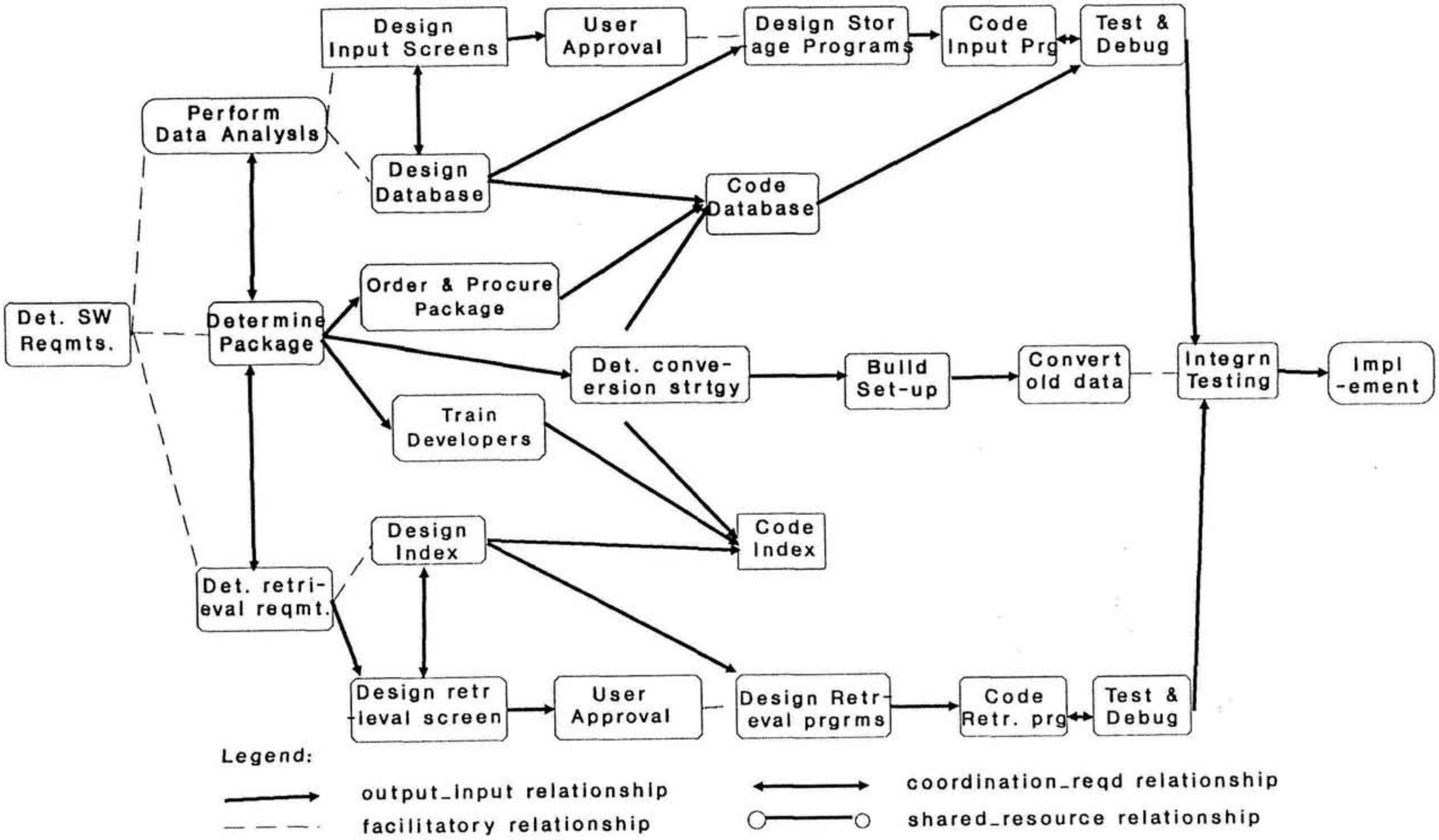

Figure 7: Work dependencies

4.1.1 Modularization of project knowledge

Given that a knowledge base can be constructed to capture the preceding information, our purpose now is to develop a methodology for partitioning this knowledge base into minimally coupled groups of design and plan decisions.

We refer to this process of forming *"islands" of project knowledge* as **normalization**. The objective of normalization is to maximize cohesiveness within each module by grouping related design objects and activities together, and to minimize coupling between modules by reducing work dependencies between such groups. Ideally, this modularization should result in knowledge partitions that are stable with respect to change; changes within one partition should not cause change in another.

The conceptual model of projects that we have developed, captures project knowledge in terms of entities and dependencies between entities. This representation is isomorphous to graph representations used extensively in Operations Research and Computer Science: project entities can be treated as graph *nodes*, and the several dependencies between them as labelled *edges*. The problem of modularizing the knowledge base is therefore one of partitioning this **knowledge graph** in a way that satisfies certain conditions.

The intuition behind the criteria for partitioning the knowledge graph comes from the following considerations: (a) it is desirable that the knowledge within each partition is *cohesive* - it should pertain typically to one design object or a family of related design objects, and (b) it is essential that the partitions are minimally *coupled* and the potential for localized control is maximized - work dependencies between partitions should be minimum.

The normalization procedure itself consists of two steps:

1. A starting partition is created by identifying "natural" design clusters - groups of activities related by conjoint and logical dependencies (*Figure 8(b)*). *Figure 8(a)* shows the first step in the simple algorithm used for this purpose.

2. The "cost" of this partition is now minimized by examining each pair of clusters, and exchanging groups of activities to minimize *work* dependencies between

clusters. Weights are assigned to the different activity-relationships to reflect the extent of dependency they cause. Coordination-required relationships have the maximum weight, followed by output-input, facilitatory, and shared-resource relationships in that order. We have adapted a well-known heuristic algorithm developed in graph-theory research for finding the minimum cost k-way partition of weighted graphs [KL70]. The result of applying our adapted version of this algorithm to the initial partitioning of the knowledge graph *(Figure 8(b))*, is shown in *Figure 9*.

Each of these final partitions is an *island of project knowledge*, and serves as the basis for modularizing the project knowledge base. In addition, these modules also serve as units for distribution of project responsibility, as seen in the discussion on the Software Project Model. The agent who is responsible for, or "owns" a module must not only be aware of dependencies within the module, but also of the extent and nature of dependence on other modules. It is conceivable that normalization may partition the project in a way that either ignores certain situation-specific dependencies between modules, or combines parts of the project that for some reason are best seperated. A post-normalization *adjustment* is therefore typically undertaken, where agents may negotiate changes in the partitioning, based on their knowledge of existing dependencies.

The end result of this process is the segmentation of the project knowledge base into modules that are weakly coupled. In *Figure 9*, these modules are shown numbered A through G. The design and plan dependencies that remain across the partition, define the *commitments* made by each module to other modules. Project control in the face of change takes the form of attempts at the level of a module, or at recursively higher levels of aggregation, to meet commitments made. In the rest of this section, we present a protocol of how this may take place.

4.1.2 Dealing with change locally

Once the project knowledge base has been modularized, and responsibility for each module assigned to different agents, the implementation of the project gets underway.

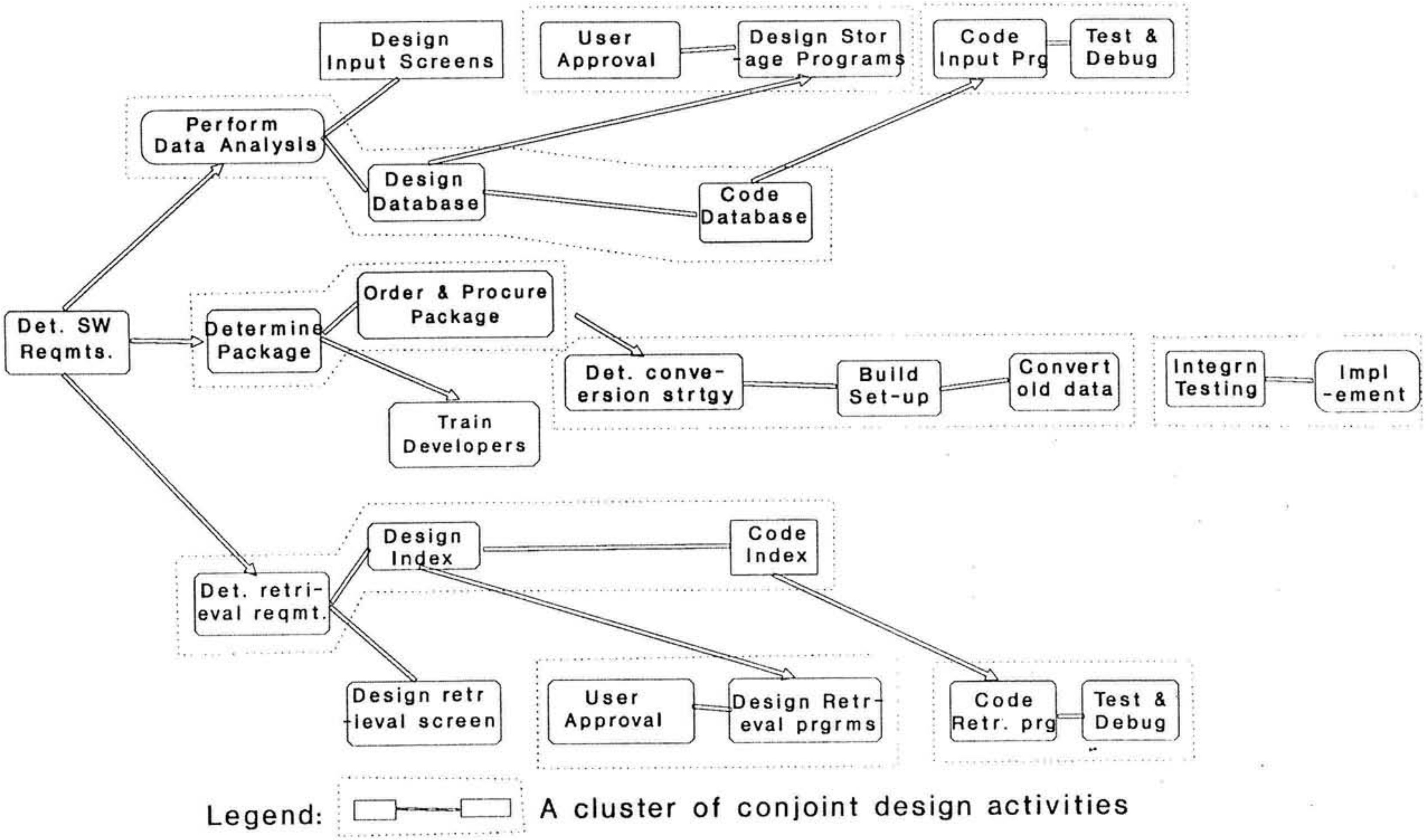

Figure 8(a): Conjoint design clusters

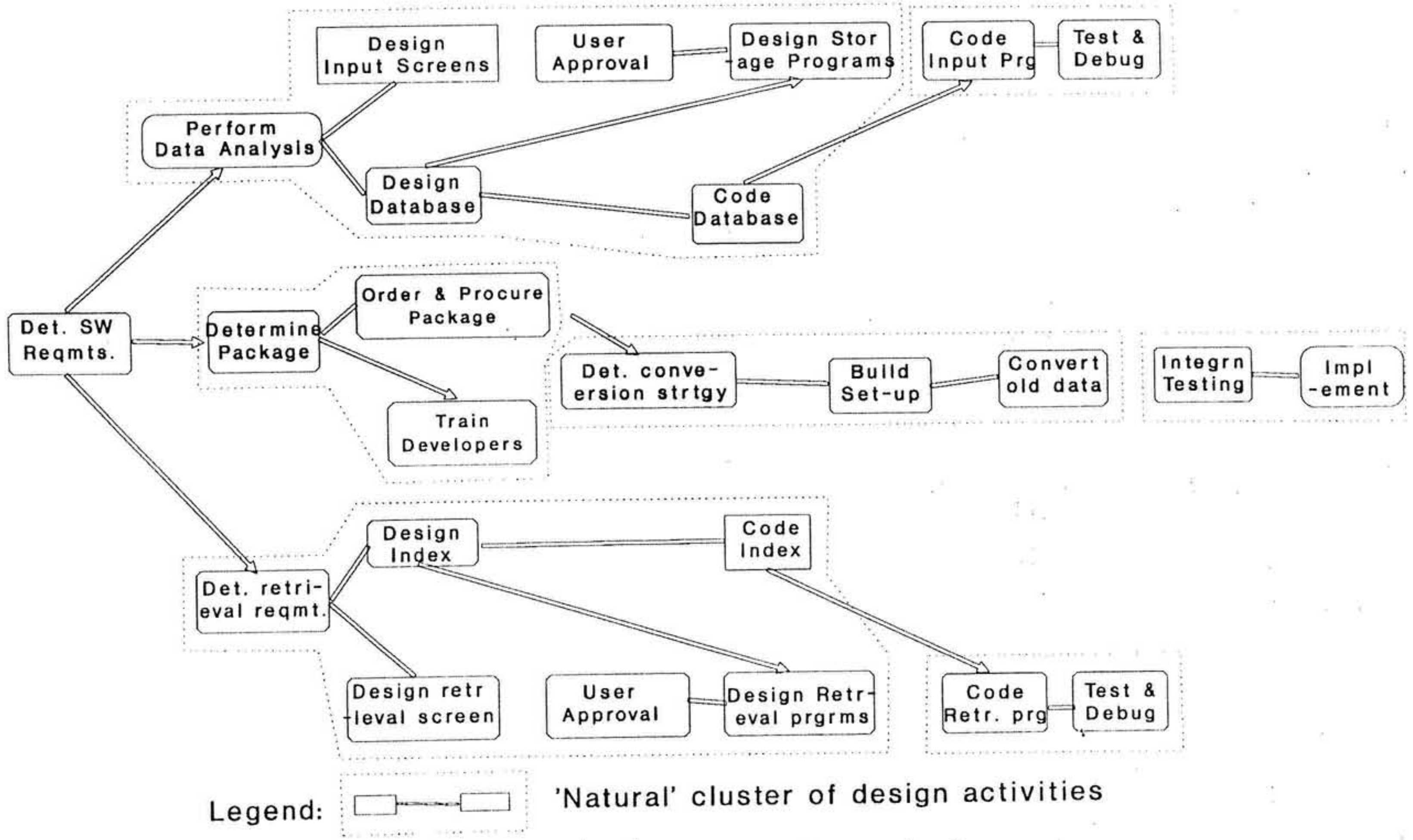

Figure 8(b): Design islands

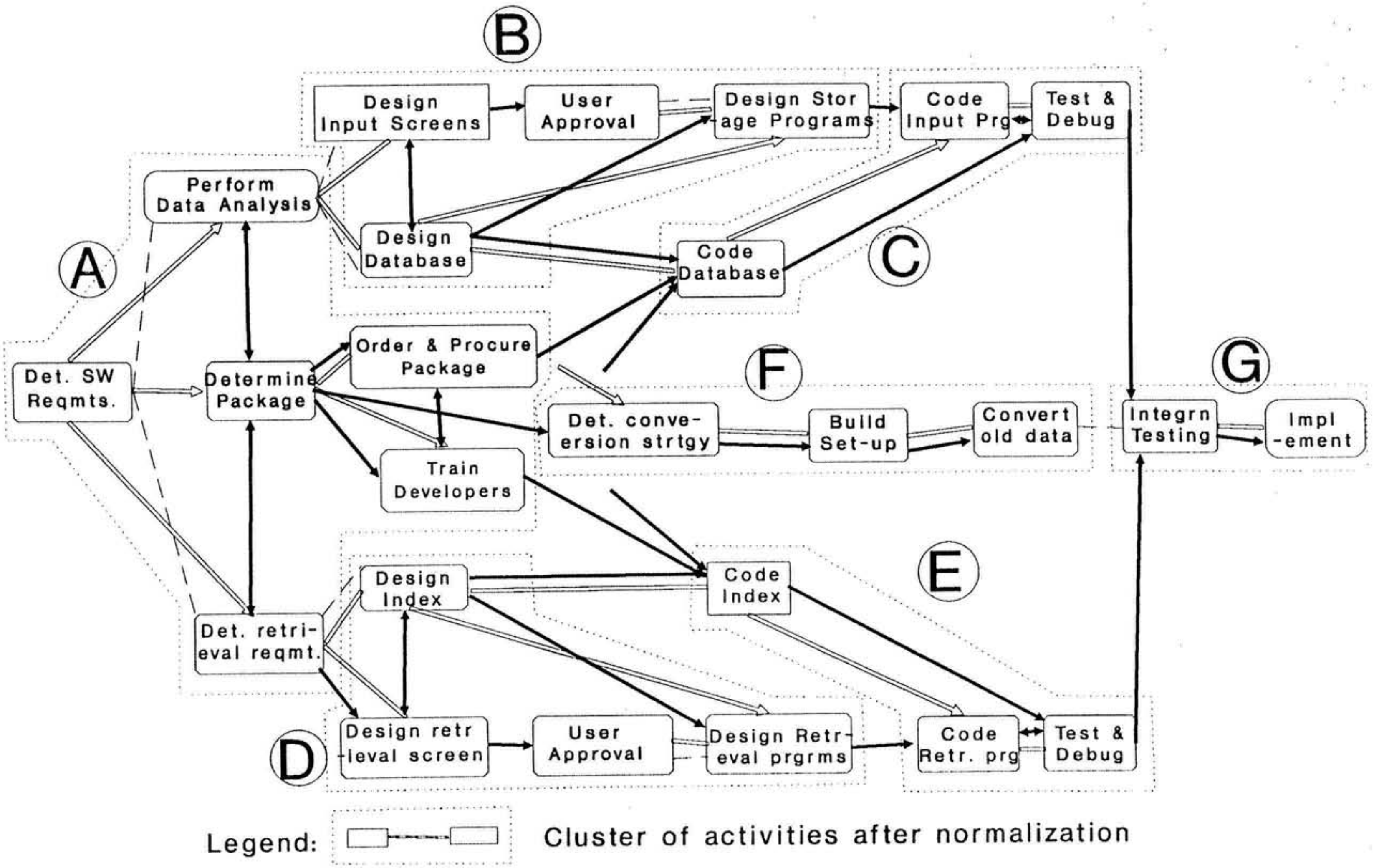

Figure 9: Islands of project knowledge

During implementation, the need for changes in design and/or plan specifications arise fairly often.

For example, external events may force the owner of *island A* (*Figure 10*), to reconsider developing the software in-house rather than procuring a package from outside and training personnel. Or alternatively, delays in building the conversion set-up (*island F* in *Figure 11*), may cause its owner to consider whether additional resources should be added, or if conversion of old data could commence before the set-up is complete.

The partitioning of the software knowledge base makes knowledge that is required for responding to changes in specifications, available locally within the affected modules in most cases. In addition, the owner of each affected module has: (a) the responsibility to meet commitments to other modules, and (b) the authority to make any changes to the design or plan, as long as their effects are localized. For instance, in the first example above, the owner of *island A* may change a design decision by electing to develop the software in-house. As a consequence, he could replace activities *Determine-Package, Order-and-Procure-Package*, and *Train Developers*, with *Develop-Package* as long as it satisfies the same design specifications. Likewise, the owner of *island F* in the second example, could decide to overlap *Build-SetUp*, and *Convert-Old-Data*, reassign resources under his control, or subcontract it out in order to meet schedule commitments to *island G*.

Sometimes, localized control either does not succeed in meeting commitments, or results in unexpected consequences. Under these circumstances, the affected *island* is in default and must communicate this condition to other *islands*.

4.1.3 Propagation of Change among Islands

Normalization does not completely eliminate dependencies between *islands*, it only attempts to reduce the incidence of certain kinds of relationships between them. From the algorithm used for partitioning the knowledge base, it may be seen that shared-resource relationships, facilitatory relationships, and sometimes output-input relationships in that order, could span activities in two *islands*. Logical dependencies, and to a lesser extent conjoint dependencies, may also exist between *islands*.

When a shared-resource relationship spans two *islands*, it represents a commitment by the first island to release the shared resource at a certain point in time, to the second island. In a similar manner, facilitatory and output-input relationships represent scheduled commitments for delivering certain outputs. A logical dependency forces related islands to commit to how certain activities will be undertaken, and conjoint dependencies indicate commitment to service a shared project objective. In summary therefore, commitments may be made about *schedules, resources, deliverables, methods* or *shared objectives.*

In the face of unanticipated events, an affected *island* may sometimes be unable to meet one or more of these commitments through localized control. It must then communicate this condition to its affected neighbors.

- When delays within an *island* cannot be controlled, schedule commitments made as a result of shared-resource, facilitatory and output-input relationships must be revised. Time delay is therefore, one kind of information flow that may be propogated across modules in the project knowledge base.

- Sometimes resources may breakdown or become unavailable. This event could affect one or more *islands* that share the resource. Changes in the availability of a shared resource, is yet another condition that may be communicated across modules.

- Activities may sometimes fail to deliver desired outputs, or accomplish desired objectives. The owner of the *island* may attempt to remedy this failure by localized replanning. Inability to do so affects commitments made about deliverables as a result of facilitatory or output-input relationships between *islands*. Changes in the status of deliverables must therefore, be communicated between knowledge base modules.

- The method chosen for performing an activity may under some circumstances, be changed. This change in the "contracted" method for an activity may call for a change in an activity in another *island* that is logically dependent on it. So changes in methods committed to, is another piece of information that may be propogated between modules.

- Last, a change in project goals or specific deliverables may sometimes be mandated by certain events. If this happens, all activities that are associated with the affected deliverable (and conjointly dependent) must be reevaluated. Changes of this nature, could also ripple through a project knowledge base.

When any of these changes occurs, and is communicated by the affected *island* to its neighbors, control of the project could take one of two forms. **First**, the neighboring *islands* could accept the change and try to accomodate it locally, or **second** they could "appeal" the change. An instance of the former may be seen in *Figure 11.* Imagine that the delay in the activity *Build-SetUp*, in *island F* cannot be controlled locally. *Island F* cannot therefore keep its schedule commitment to neighboring *island G*, and so communicates it. In this case, *island G* accepts the change in commitments after considering that completion of *Convert-Old-Data* though desirable, is not necessary for beginning *Integration-Testing.*

4.1.4 Formation of Coalitions for Project Control

When an *island* indicates that it cannot accept a change in commitments and appeals it, a mini-breakdown in the project is signalled. At this point, a higher-level aggregate *island* is created by forming a coalition between the *island* in default and the *island* on appeal, and an attempt is made to control the change within this aggregate.

Once a coalition is formed, the corresponding knowledge base modules are combined into one so that all available knowledge may be used for plan or design revision. Localized reevaluation of the project is undertaken within this aggregate *island* to ensure that inconsistencies and redundancies are eliminated. Plan revision by rescheduling or reallocation of resources, and design revision within the aggregate *island*, may then be made in order to meet commitments.

Examples of this are shown in *Figures 10 & 11.* Imagine that halfway through the project, a design decision is made to implement the system using a relational rather than a hierarchical database as determined earlier. The database schema that will be delivered by *Design-Database* in *island B* will now be in a different format from that committed to *island C*. If *island C* appeals this change, a higher-level coalition is formed by combining B and C (*Figure 10*). Combining the corresponding knowledge base modules provides access to all information required for incorporating this change in project design specification. Likewise, *Figure 11* illustrates the case when Systems Analyst I responsible for *Det-Retrieval-Requirements* (and also *Design-Retrieval-Screen*), falls ill. *Island A* cannot keep its schedule and resource commitments to *island D*, which appeals this change. A higher-level coalition, *island A+D* is formed and the project plan is revised locally. The other systems analyst available in *island A* is reallocated to *Det-Retrieval-Requirements* and *Design-Retrieval-Screen*, while *Design-Index* proceeds as per schedule.

This method of adaptive, need-based formation of coalitions between *islands*, results in a more flexible approach to management of change. It allows us to control each part of the project independently until the need arises for considering interactions. Even then, these interactions are not dealt with in a strictly hierarchical manner as advocated in traditional control paradigms, but in a more oppurtunistic and need-based manner that minimizes the effects of the disturbance.

In summary therefore, this section offered a methodology for managing change in relatively ill-structured project domains. We have argued that in such domains a knowledge base that captures the 'complete picture' facilitates pro-active project planning and interactive project control. We outlined a technique for pro-active planning, that normalizes project knowledge bases by partitioning the knowledge graph into minimally dependent *islands*. A protocol for interactive change management was advocated, that recursively attempts localized control within *islands* or higher-level aggregates. When local control fails, changes in commitments are communicated to neighboring *islands*, and if needed, coalitions are formed between affected *islands* to manage the change.

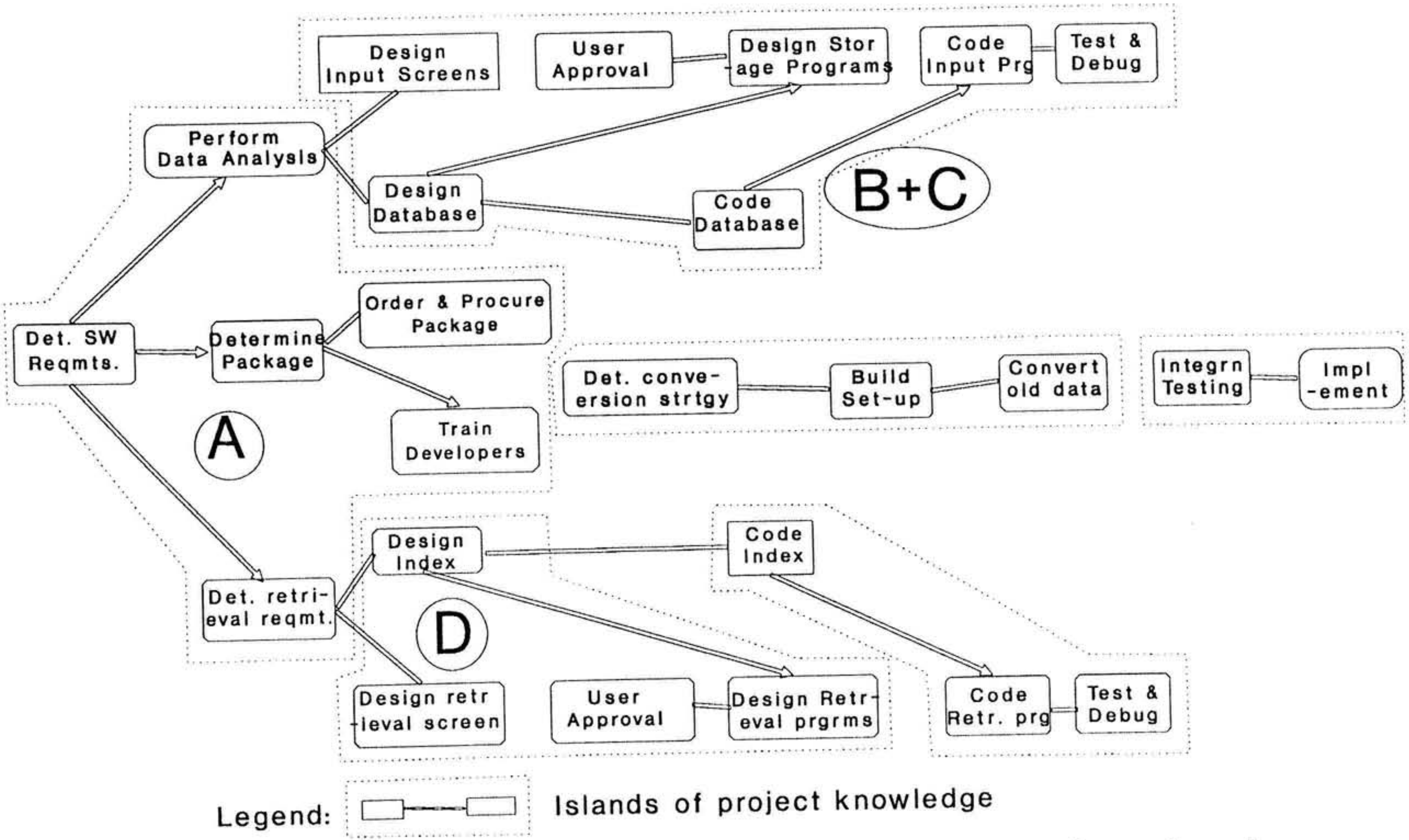

Figure 10: Responding to changes in design

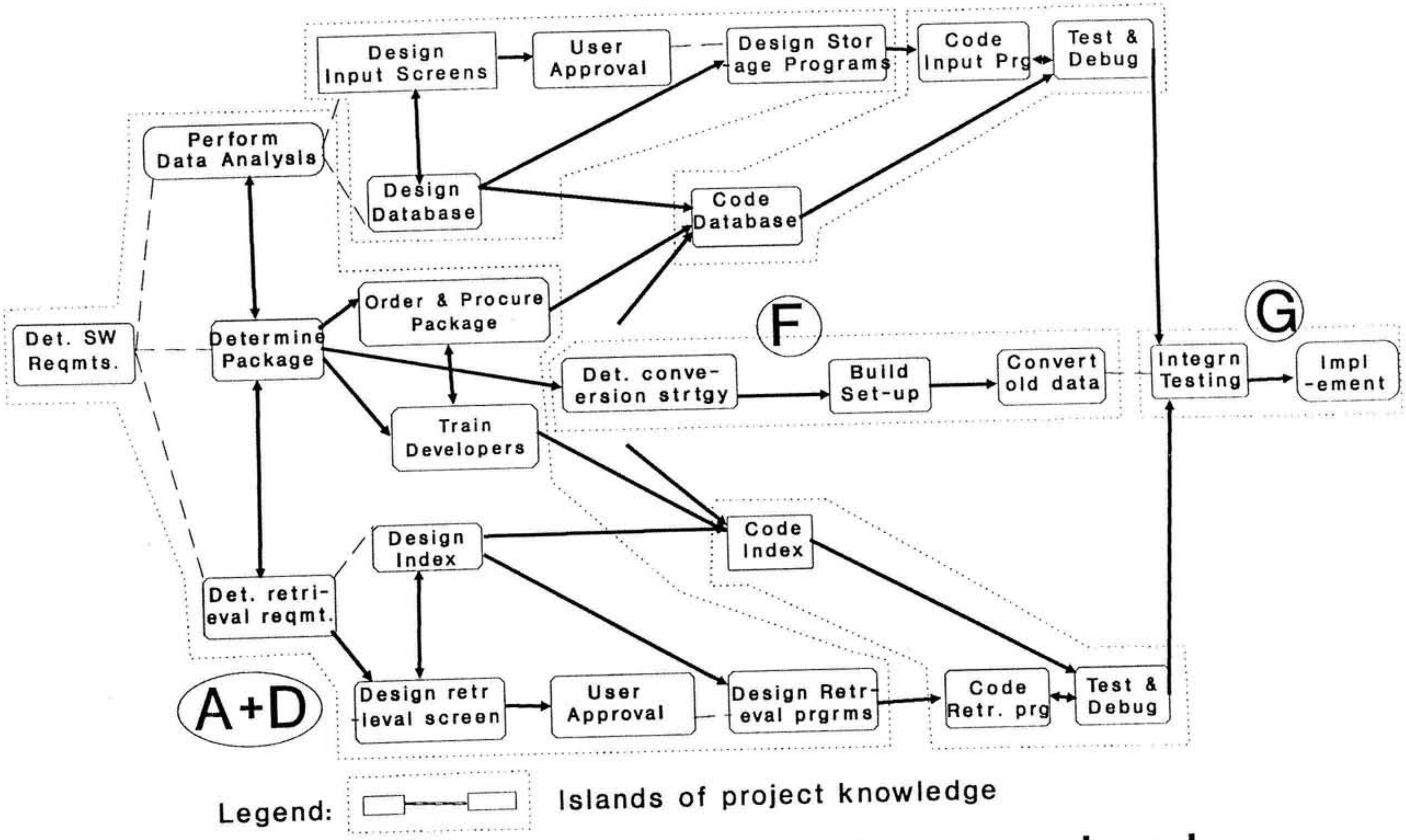

Figure 11: Responding to changes in plan

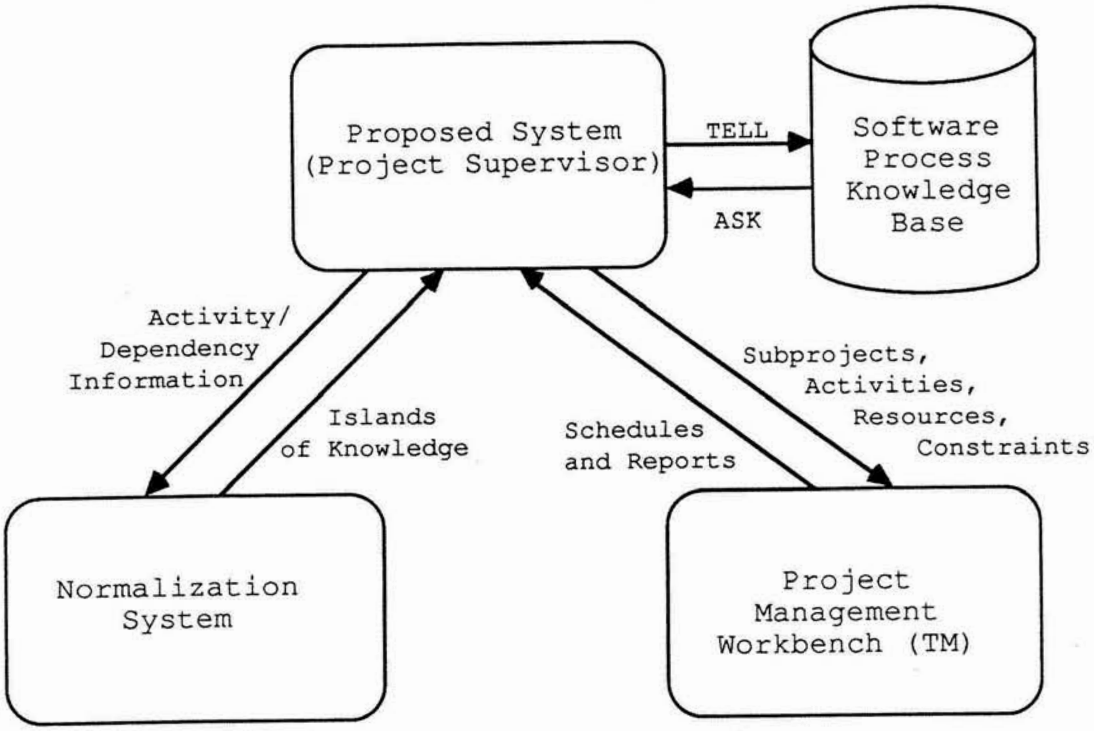

Fig. 12: Implementation Architecture

mnmn

4.2 Implementation and Evaluation

This methodology is currently being implemented on a microcomputer based system, for purposes of laboratory and field evaluation. In this subsection, we outline the architecture and construction of the system, and the design of a computer-based simulation study that will be used to for evaluating its effectiveness.

4.2.1 Implementation of the System

The implementation has been designed to exploit the fact that our project model is a 'consistent extension' of the classical project model. Our objective in building the system is to create a prototype that will allow users to pro-actively plan a software development project, representing information in terms of the enhanced project model. The system will then recommend a partitioning of the project knowledge into *islands*. After adjustments are made, it will (i) generate a schedule for each *island* and the project as a whole, using traditional scheduling algorithms, (ii) document the commitments that are made by each *island* to other *islands*.

The system architecture is shown in *Figure 12*. It consists of 3 modules: a knowledge base cum supervisor module, a normalization module, and a scheduling cum resource allocation module.

The knowledge base cum supervisor module controls interaction with the user. It allows users to plan a project by specifying objectives/deliverables, activities, resources, and the dependencies they consider in making design and plan decisions. When the user has finished representing project-related information, he or she may initiate normalization of the project knowledge base. This module is currently being implemented in Smalltalk V/286, an object-oriented programming environment for microcomputers.

The normalization module is a FORTRAN implementation of the algorithm that we outlined for partitioning the knowledge graph. The scheduling cum resource allocation module is an off-the-shelf project management package. It is primarily used for applying powerful scheduling algorithms, and providing the user with sophisticated graphing and report generation capabilities.

The implementation as a whole, falls under the category of what are being increasingly referred to as "multi-paradigm" systems. These are software applications which consist of several modules, each written in a language that is best suited for its given function; inter-module communication of data and control flags occurs through the use of shared data files.

4.2.2 Empirical Evaluation

The purpose of the implementation outlined above was to provide a usable prototype for field evaluation, and a platform for conducting laboratory studies.

A laboratory study is currently being planned that seeks to evaluate the effectiveness of the interactive change management methodology we have developed. In particular, we are interested in its effects on the extent to which a project's designs and plans are affected by disturbances, and the overall cost of responding to such change.

Using a computer-based simulation, disturbances such as changes in deliverables or design specifications, time overruns, and resource breakdowns will be generated. The responses of interactive change management and classical project control, to identical disturbances will also be simulated for a variety of project networks. A log of any revisions in design or plan will be maintained electronically, and project completion times and costs documented.

A comparison of the change logs will be made to obtain qualitative insights into differences between the two methods. The vector of changes distributed across each of the *islands* will provide a metric for the extent of disturbance propogation. Statistics of completion times and costs can likewise be determined to compare the overall costs incurred during change management.

The setup for this simulation study is currently under development and will be ready shortly. Through this study, we expect to demonstrate that the methodology we have developed results in significantly reduced propogation of disturbances across the project knowledge base. We also expect to show that cost efficiencies are comparable, if not better than classical approaches to project control.

5 Summary and Outlook

The starting point of this paper was the observation that, in large, ill-structured projects, change of requirements and available resources is so frequent that rigid project planning techniques become obsolete. Consequently, our "islands of project knowledge" approach attempts to help project managers plan pro-actively, with the goal of organizing projects so that most change can be dealt with locally. Knowledge representation techniques were then used to represent such change-friendly project structures, simultaneously supporting actual content-oriented project work. Given the current limitations of such systems, a project management methodology and implementation approach were proposed that start from currently available project management software and just enhance it with a few crucial qualitative features.

Both the underlying model and the actual implementation have been, or are in the process of being, empirically validated in laboratory and field settings. In this way, we hope to improve management performance on a class of important projects that have enjoyed little effective support by information system technology so far, and have consequently been plagued by time and cost overruns continuously.

A distinctive feature of our approach is that the distribution of project control shifts responsibility back to human collaborators, rather than just treating them as resources, hopefully fostering creativity and responsiveness. To achieve this goal, models and tools for project decomposition and re-integration as presented in this paper are, of course, only a first step. Further work on actual group support tools is therefore underway that facilitate idea generation and organization, negotiation, and commitment monitoring.

References

ACK81 Ackoff, R.L. (1981). *Creating the Corporate Future*, New York: John Wiley.

ALL83 Allen, J.F. (1983). Towards a general theory of action and time. *Artificial Intelligence 23*, 2, 123-154.

AM88 Abdel-Hamid, T.K., Madnick, S.E. (1988). Lessons learned from modelling the dynamics of software development. WP 2069-88, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Mass.

AS84 Attardi, G., Simi, M. (1984). Metalanguage and reasoning across viewpoints. *Proc. ECAI '84*, Pisa, Italy (North-Holland), 315-324.

BB88 Bimson, K.D., Burris, L.B. (1982). Conceptual model-based reasoning for knowledge-based software project management. *Proc. 21st Hawaii Intl. Conf. System Sciences*, Kona, HW, 255-265.

BCEGRS86 Begeman, M., Cook, P., Ellis, C., Graf, M., Rein, G., Smith, T. (1986). Project NICK: meetings augmentation and analysis. *Proc. First Intl. Conf. Computer-Supported Cooperative Work*, Austin, TX, 1-6.

BMR89 Bartusch, M., Moehring, R.H., Radermacher, F.J. (1989). Design aspects of an advanced model-oriented DSS for scheduling problems in civil engineering. *Decision Support Systems 5*, 3 (this volume).

BRO82 Brooks, F.P. (1982). *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, Reading, MA.

COORD86 COORDINATOR, (1986). *The COORDINATOR Workgroup Productivity System: Workbook and Tutorial Guide.* Action Technologies Inc., San Francisco CA.

DAIDA88 Jarke, M., DAIDA Team (1988). The DAIDA environment for knowledge-based information systems development. *Proc. ESPRIT '88: Putting the Technology to Use*, Brussels, Belgium, 405-422.

DDSVZ86 DeCindio, F., G. De Michelis, C. Simone, R. Vassallo, and A. Zanaboni. (1986). CHAOS as a coordination technology, *Proc. First Intl. Conf. Computer-Supported Cooperative Work*, Austin, TX, 325-340.

ES84 Ericsson, K.A. and H.A. Simon. (1984). *Protocol Analysis: Verbal Reports as Data.* Bradford Books/MIT Press, Cambridge, MA, 1984.

FIK82 Fikes, R.E. (1982). A commitment-based framework for describing informal cooperative work. *Cognitive Science*, 6():331–347.

GAL73 Galbraith, J.R. (1973). *Designing Complex Organizations. Addison-Wesley series on Organizational Development*, Addison-Wesley, Reading, MA.

GJLP87 Gilham, L.M., Juellig, R., Ladkin, P., Polak, W. (1987). Knowledge-based software project management. Report KES.U.87.3, Kestrel Institute, Palo Alto, Ca.

HAHN89 Hahn, U. (1989). Dialogstrukturen in Gruppendiskussionen. Report, DFG Project Ja-445/1-1, University of Passau, W. Germany. Submitted for publication.

HJ89 Hahn, U., Jarke, M. (1989). CoNeX: Coordination and negotiation support for expert teams in project management. *European Conference on Computer-Supported Cooperative Work*, London, UK.

HJKFP Hahn, U., Jarke, M., Kreplin, K., Farusi, M., Pimpinelli, F. (1989). CoAUTHOR: a cooperative group authoring environment. Report, ESPRIT Technology Integration Project 2105 (MULTIWORKS), *European Conference on Computer-Supported Cooperative Work*, London, UK.

HOF88 Hofbauer, T. (1988). IPMSS, an intelligent project management support system. Diploma thesis, Technical University of Munich, W. Germany.

JDL88 Juellig, R.K., Daum, M., Ladkin, P.B. (1988). Approaches to planning the the Project Management Assistant. *Proc. 3rd Annual KBSA Conf.*, Utica, N.Y., 31-51.

JJR88 Jarke, M., Jeusfeld, M., Rose, T. (1988). A global KBMS for database software evolution: documentation of first ConceptBase prototype. Report MIP-8819, University of Passau, W. Germany.

JJR89 Jarke, M., Jeusfeld, M., Rose, T. (1989). A software process data model for knowledge engineering in information systems. *Information Systems*, to appear; also available as Report MIP-8910, University of Passau, W. Germany.

KED84 Kedzierski, B.I. (1984). Knowledge-based project management and communication support in a system development environment. *Proc. 4th Jerusalem Conf. Information Technology*, Jerusalem, Israel.

KL70 Kernighan, B.W., Lin, S. (1970). An efficient heuristic procedure for partitioning graphs, *The Bell Systems Technical Journal*, February: 291-307.

KMSB89 Koubarakis, M., Mylopoulos, J., Stanley, M., Borgida, A. (1989). Telos: features and formalization. Technical Report KRR-4, Dept. Computer Science, University of Toronto, Canada.

KP87 Kurbel, K., Pietsch, W. (1987). Projektmanagement bei Expertensystem-Entwicklungen. Report no. 12, Lehrstuhl fuer Betriebsinformatik, University of Dortmund, W. Germany.

MOS85 Mostow, J. (1985). Towards better models of the design process. *AI Magazine 6*, 1, 44-57.

SFG85 Sathi, A., Fox, M.S., Greenberg, M. (1985). Representation of activity knowledge for project management. Report CMU-RI-TR-85-17, Carnegie-Mellon University, Pittsburgh, Pa.

SIM62 Simon, H.A. (1962). The architecture of complexity. *Proceedings of the American Philosophical Society*, 106:467–482.

SRI89 Srikanth, R. (1989). Islands of control: a knowledge-based approach for managing change in projects. Ph.D. dissertation, Leonard N. Stern School of Business, New York University, New York, NY, *in progress.*

THO67 Thompson, J.D. (1967). *Organizations in Action.* McGraw-Hill, New York NY.

VAU88 Vauquois, P. (1988). PIMS, a Project Integrated Management System. *Proc. ESPRIT '88: Putting the Technology to Use*, Brussels, Belgium, 392-404.

WA88 Wile, D.S., Allard, D.G. (1988). Worlds: aggregates for object bases. USC Information Sciences Institute, Marina del Rey, Ca.

WF86 Winograd, T. and F. Flores. (1986). *Understanding Computers and Cognition: A New Foundation for Design.* Ablex Publishing, Norwood NJ.

YC78 Yourdon, E. and L.L. Constantine. (1978). *Structured Design.* Yourdon Press.