NASA-CR-193289

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

DEVELOPMENT OF A CASE TOOL TO SUPPORT DECISION BASED SOFTWARE DEVELOPMENT

By

Christian J. Wild, Principal Investigator

Annual Progress Report
For the period ended March 31, 1993

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-0001

Under
Research Grant NAG-1-1426
Dr. Dave E. Eckhardt Jr., Technical Monitor
ISD-Systems Architecture Branch

July 1993

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

DEVELOPMENT OF A CASE TOOL TO SUPPORT DECISION BASED SOFTWARE DEVELOPMENT

By

Christian J. Wild, Principal Investigator

Annual Progress Report
For the period ended March 31, 1993

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-0001

Under
Research Grant NAG-1-1426
Dr. Dave E. Eckhardt Jr., Technical Monitor
ISD-Systems Architecture Branch

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508-0369

July 1993

Table of Contents

1. Introduction

This report presents a summary of the accomplishments of this research group over the past one year.

In accordance with our proposal, the achievements of this period of time are:

– made demonstrations with DHC, a prototype supporting DBSD methodology, for Paramax personnel at ODU; met with Paramax personnel on a regular basis to discuss DBSD issues, the process of integrating DBSD and Refinery and the porting process model (see also Attachment 3).

– completed and submitted a paper describing DBSD paradigm to IFIP '92, Spain, which was accepted and presented; completed and presented a paper describing our approach for software reuse at the Software Reuse Workshop held in April 93 in Washington, D.C.(see also Attachment 1)

– continued to extend DHC with a project agenda, facility necessary for a better project management.

– completed a primary draft of the re–engineering process model for porting, defined at the requirements level for Paramax re–engineering problem(see attachment 2 and 6).

– created a logging form to trace all the activities envolved in the process of solving the reengineering problem (see also Attachment 7).

– according to our discussions with the Paramax personnel we have developed a primary chart with the problems envolved by the reengineering process (see Attachments 3, 4, 5).

2. Meetings with Paramax personnel

In this period of time we have met with Paramax personnel on a regular basis to discuss DBSD issues, the process of integrating DBSD and Refinery and the porting process model and we made demonstrations for them with DHC, a prototype supporting DBSD methodology (see also Attachment 3). Also, Tammy Taylor (Paramax) demonstrated Refinery for the ODU team.

3. Paper describing DBSD paradigm

We completed and submitted a paper describing DBSD paradigm to IFIP '92 Conference, Spain, which was accepted and presented. We also completed and presented a paper describing our approach for software reusability at the Software Reuse Workshop held in April '93 in Washington, D.C.(see also Attachment 1).

4. Issues in solving the software re–engineering problem

4.1. The Re–engineering Problem

Paramax desires a reengineering capability to address the following scenarious:
– port applications from one machine to another.
– translate an existing application on a source machine to an application in a new language, machine independent so that they can have a kernel version of an application which runs on several platforms.
– enhance the features of an application, extract the best features from several versions of a given software system and create one kernel version.

They will need support for both porting and reverse porting processes since they want to keep consistent the 2 versions of a program that has been ported.

4.2. Approach

4.2.1. Alternatives

To address these problems we can adopt one of the following alternatives:
– build solution from scratch, develop a fully automated, noninteractive system for specic cases.
– develop an expert system using only DHC.
– integrate DBSD and Refinery into a Unified Environment.
– adopt some other non–automated tools suited to our purposes(like UNIX "awk", for example).

We have decided to adopt the alternative of integrating DBSD and Refinery, since in house and local capability exists and Refinery may allow for significant automation through use of transformation rules, DBSD creating the environment for recording the methodology of the processes performed and the decisions taken during the development of these processes.

For this integration of DBSD and Refinery we see two possibilities:
– make them loosely coupled, seeing each other like a black box that executes its job sequential in time with respect to the other tool.
– make them tightly coupled, i.e. embed DHC in Refinery or viceversa. For this case we need a deep understanding of the source code, capabilities and functions of Refinery. We would like to have answers from Paramax to the following questions that would help us to figure out the final approach of this problem:
– which are the capabilities and functions of Refine? Are they noninteractive so they can be wrapped into DHC functions, are they sufficiently small to be embedded?
– how is the precision of decision structure maintained through the Refinery transformation?

Up to now we have developed the loosely coupled version, choosing the simple way of porting, in order to better understand the process.

For example, if we want to port an application from Harris to Honeywell or Microfocus Cobol, or Honeywell to Harris or Microfocus Cobol, one should perform the following steps:

- select sample Harris code
- select functionally identical Honeywell code
- parse Harris code into Refine object base, generating an Abstract Syntax Tree(AST1)
- parse Honeywell code into Refine object base, generating an Abstract Syntax Tree(AST2)
- perform manual comparison on programs and components
- isolate problem areas:
  - areas where there is no one–to–one transfer
  - overlapping of functional modularity.
- develop Refine code to convert Harris Cobol to Honeywell Cobol
- develop Refine code to convert Honeywell Cobol to Harris Cobol
- develop Refine code to convert Harris Cobol to Microfocus Cobol
- develop Refine code to convert Honeywell Cobol to Microfocus Cobol
- perform manual completion of the conversion in each of the above cases
- use DBSD for recording methodology and the decisions taken during the development process.

In the development process one might find decision views attached to the original source, decision views attached to the transformation rules or decision views attached to the target source. The latest might include decision views attached to the original source, decision views attached to untransfered code, decision views attached to new code generated during transformation process, mapping of decision views attached to transformation rules to the target code. In order to be able to distinguish among this diversity of decision views we need to develop in DHC a mechanism for "filtering" the views.

4.2.2. Upgrading DHC

In developing the solutions for the reengineering problem we began to recognize the importance of managing problems which are in a state of transition. Problems which have not been fully integrated into the document base must be kept visible. A project agenda is used to record the state of all problems under active development. This project agenda may contain tentative alternate solutions to problems from which an assessment can be made. Once a commitment is made, the chosen solution can be linked into the document base. So, problems which have been identified but not yet solved are kept in the project agenda. Project managers use the information in the project agenda to

allocate resources to solve these problems. Focusing on this process instead of the products of development allows management to control the scheduling of activities.

4.2.3. The chart of the reengineering problem space

According to our discussions with the Paramax personnel we have developped a primary chart with the problems envolved by the reengineering process (see Attachments 3, 4, 5). It contains the graphical equivalent of the problem and decision spaces. The problems in the chart are indexed by the number in the DHC.pd file (the problem description file). It includes the problems, alternatives and output of the problems, linked together by decision, justification, alternate and output links.

4.3. Process Model

In the referred period of time we have concentrated our efforts in creating a primary draft of the re–engineering process model for the porting problem, in order to better understand it. We have decided to write separate methodologies for porting, enhancing and translating because the process is too little understood to fully develop a general methodology for everything. This process model addresses the activities and their input and output used in creating and using the information necessary to the problem solving. By writing an explicit process model, the roles of different members of the software engineering team and management is clarified. In addition, it is possible to identify desirable functionality for the software engineering environment.
We have attached the primary draft of the porting problem (Attachment 2) and the DBSD process model for the traditional life cycle (Attachment 6), as well as the notations used in these two process models.

4.4. Evaluation Process

In order to evaluate the conditional decisions during the process of solving the porting problem we have created a logging form that will help us to keep a notebook of activities, tools used, features applied, the time spent in each activity as well as the products of each activity (see Attachment 7).
All this information will be applied to a statistical analizer that will help us to figure out the frequency , duration of each feature, the size of various parts of documentation(decisions, transformation rules, source code, BNF rules, etc.), amount of new parts vs. changes in existing ones in the dynamic process of this problem solving.

4.5. Market Study done at Paramax

In order to complete our research and document preparation, we need answers to several questions which are listed below. These questions represent concerns about the market which Paramax is serving and expected market conditions for the future. This will assist us in defining our reengineering model and associated activities. We request Tammy

Taylor from Paramax/Virginia Beach to gather the answers to these questions or to put us in touch with the appropriate personnel that can provide us the guidance necessary to proceed.

1. Does Paramax/Virginia Beach and Paramax/Corporate have contracts for porting software from one platform to another? If so, how many and to what extent? For example, a. How big are the contracts to port in terms of lines of code? b. What language are they in? c. What is the value of the contracts? d. What solutions to these problems have been used in the past? e. What problems have been encountered with these solutions?

2. In porting software, is the requirement a one to one port? Meaning, are you to just move the software as it stands and provide the necessary mechanism for running in another environment? If so, once the port is complete, will you be contracted to enhance ported software in the new environment? If not, what is the expected duration of the ported software on the new machine? (i.e. will it be redeveloped from scratch in the new environment? etc.?)

3. How often do you foresee performing software porting? software translationg? and software enhancements in the foreseeable future? Anotherwords, what is your predicted bussiness direction?

4. What impact does the Department of Defense (DOD) have in your business goals?

5. What does the DOD mean by reuse/reengineering? Are you solely driven by the DOD requirements?

6. What are the characteristics of reuse?

7. What are the characteristics of reengineering?

8. What types of software systems do you envision the DOD wanting to perform reuse/reengineering on?

9. What are the strategies of DOD for the foreseeable future?

10. What relation does ICASE play in those strategies?

ATTACHMENT #1

Paper and viewgraphs

describing DBSD paradigm

Software Life Cycle Support - Decision Based Software Development

Chris Wild and Kurt Maly

Department of Computer Science,
Old Dominion University, Norfolk, VA 23529-0162 USA

Abstract

The software engineering life cycle encompasses a broad range of activities from the initial elicitation of the system requirements to the continuing evolution of the operational system. These activities can be best supported if there is a unifying paradigm which can integrate functional and non-functional problem-solving, process management, and knowledge acquisition and reuse. The Decision Based Software Development (DBSD) paradigm structures the software development and evolution process as a continuous problem-solving and decision making activity. In the DBSD paradigm, the software engineering team identifies and articulates software development problems, proposes alternative solutions, develops supporting justifications from which a decision is made. By making the problem solving process visible, DBSD allows management to control the creative, and sometimes chaotic, set of activities comprising software development. By recording the decisions, decision making rationale and the relationships among decisions and between decisions and the products of software development, the source code and related documents are structured significantly different from traditional structures such as the modular or data flow view. This structure associates a decision with only those parts of the documents affected by that decision. These decision views support continued evolution of the software system because both the rationale for individual decisions are recorded as well as the interrelationships among decisions. Documenting these interrelationships helps the software engineer assess the impact of changing a decision and to understand the consistency requirements among a set of decisions.

Keyword Codes: D.2.0; D.2.2; D.2.7; D.2.9
Keywords: Software Process; Software Maintenance; Engineering Design

1 Software Development Problem

The engineering of large systems is a complex undertaking which requires sound technical methods and careful project management. In order to make significant progress in improving software creation and maintenance, a better understanding of, and support for, the development process will be necessary. By focusing on the software products most approaches to software development do not adequately support the design process itself. Non-functional requirements are poorly represented by the current software engineering structures. We believe that supporting the design process will require a fundamentally different development paradigm which addresses the entire systems development life cycle. In this paper, we describe the Decision Based Software Development (DBSD) paradigm for systems life cycle support.

A good software engineering methodology should provide knowledge sources, reasoning agents and process management. The purpose of documentation is to capture knowledge for later reuse. In

addition, project knowledge is contained in the minds of the software engineering team. Reasoning is carried by the members of the software engineering team. This reasoning can be assisted and in some cases replaced by software engineering tools. The process describes how knowledge is created and used during the development effort. It deals with the management of the knowledge and reasoning resources.

It is our view that automating the project knowledge base will provide the greatest near term leverage for addressing software engineering problems. Existing document bases suffer from two major deficiencies. First, not all the critical information is contained in the document base. Traditional documents typical only record the results of the problem solving process in terms of the particular solutions taken and only if this solution results in a visible product or deliverable. Much of the understanding and justifications for making particular decisions exists only in the minds of the original developer. This information consists of relationships between different requirements which require tradeoffs to be made, promising development paths that later proved to be dead-ends and alternative paths that could lead to a better solution if more resources were available to pursue them. The person who subsequently uses the document base to make changes, must be able to reverse engineer this missing information. The second problem is that the organization of the document base may not support easy access to relevant information that is contained there as it is needed. At the very least, more effort is expended to locate this information. In the worst case, this information must be reverse engineered. Reverse engineering critical information is both time consuming and error prone and contributes significantly to the software problem.

The content and organization of the project document base thus plays a critical role both in guiding the initial development and supporting system evolution. During the process of software development the software engineer needs access to only a selected subset of information from the document base. The subset of information needed to perform a particular task is called the **closure** of that task. The **ideal closure** contains exactly the information the software engineer requires to perform the task. The **actual closure** refers to that subset of information which the software engineer can retrieve from a particular document base. The actual closure is determined by the organization of the document base, the set of information retrieval tools available, and the process used to build the closure. Much of the effort on structured programming has been to organize the program text and development process to support closure based on modular structure and functional abstraction.

The document base should be used to support the development process. This implies that the document base contain more than the final products of each phase of the life cycle. In order to support the early phases of development, the document base must record the problem solving process, even if the relationships between these problems and the final products is not yet clear. The development process must allow for easy evolution and maintenance of the document base. In addition the documentation effort should not place an undue burden on the developers. Information that is easy to reverse engineer, need not be included in the document base. It should be easy to add missing documentation and to correct the inconsistency of existing information.

2 A New Approach: Decision Based Software Development

The Decision Based Software Development(DBSD) Paradigm has been proposed to support the process of software creation and evolution. DBSD belongs to the school of thought that the development process can be modeled as a set of related problem solving episodes [?, ?, ?, ?].

Organizing the document base and the development process around problem solving and decision making is more general than using data flow, top down design, module decomposition, object oriented design or other structured methods. Problem solving is a universal activity which spans the life cycle. For example, many of the decisions in determining the requirements are based on weak justifications or are made without a full understanding of their consequences. Documenting the decision made during the requirements definition would help the designers understand which requirements are firm and which could be revised in order to build a better or cheaper design.

Example 1 *A system requirement stipulates a response time under three seconds for 90% of the transactions under worst case loading. Providing the resources to meet this requirement may be wasteful if the worst case load occurs extremely rarely or if the three second limit was chosen arbitrarily.*

Both functional and non-functional requirements involve problem solving. Non-functional problem solving receives very little attention in most software development approaches. Many of the solutions to non-functional problems do not directly result in software products but the provide the constraints under which software products are developed. The response time requirement described in the above example may be the proposed solution to the problem of keeping the user attention and reducing frustration, but it does not in itself produce code. It does, however, affect the choice of data structures, algorithms, overall software and hardware architecture and hardware performance requirements. Additionally, problem solving is process oriented. Since engineering involves many tradeoffs, there will be times when earlier decisions will have to be changed. Documenting the effects of these decisions will ease the burden of making these changes both during initial development and system evolution.

Recording the decisions and structuring the document base by these decisions has an additional advantage. In initial development, focusing on the identification of problems, alternate solutions and justifications provides structure into what is a creative but often unstructured process. The status of the progress among all problems is recorded. If a problem has been solved, the decision and its justification is recorded together with all its effects. Problems which have been identified but not yet solved are kept in the project agenda. Project managers use the information in the project agenda to allocate resources to solve these problems. Focusing on this process instead of the products of development allows management to control the scheduling of activities.

The primary advantage to recording the decision structure of a project comes forth during software maintenance. If the software contains a fault it is because at some stage in the problem solving process a faulty solution was chosen. If a solution leads to poor performance, then the decision which choose that solution must be revisited and an alternate solution should be chosen. When adding new functions to a system, the documentation of decisions will help the software engineer decide which solutions can be reused. Another advantage of relating documentation to decisions is when decisions are changed, those parts of the document base which are affected by that decision are more easily accessed and updated. This helps address the problem of keeping the documentation up to date.

The problem solving model is in contrast to the transformational model of development in which an initial abstract document describing the system is transformed into an efficient operational system. One of the problems not addressed in transformational system is how the initial system description is generated. The basic problem solving process involves the following steps:

1) **Identification and articulation of the problem to be solved.**

2) **Generation of alternate potential solutions.**

3) **Validation and Justification.** Before a proposed solution can be adopted, it must be validated as a feasible solution and its choice among all the feasible solutions must be justified. This justification is done in the context of other decisions and constraints.

4) **Commitment.** If there are several alternate solutions, then a decision must be made. This decision is justified both by the quality of the solution and by the context in which the decision is made.

Because a solution to a problem may itself constitute a subproblem, the process of problem solving is recursive.

DBSD introduces new information structures into the project document base as shown in Fig. ??. The decisions are listed on top with the links to the relevant problems visible. In this figure we have organized the document base according to the traditional life cycle and hence have grouped

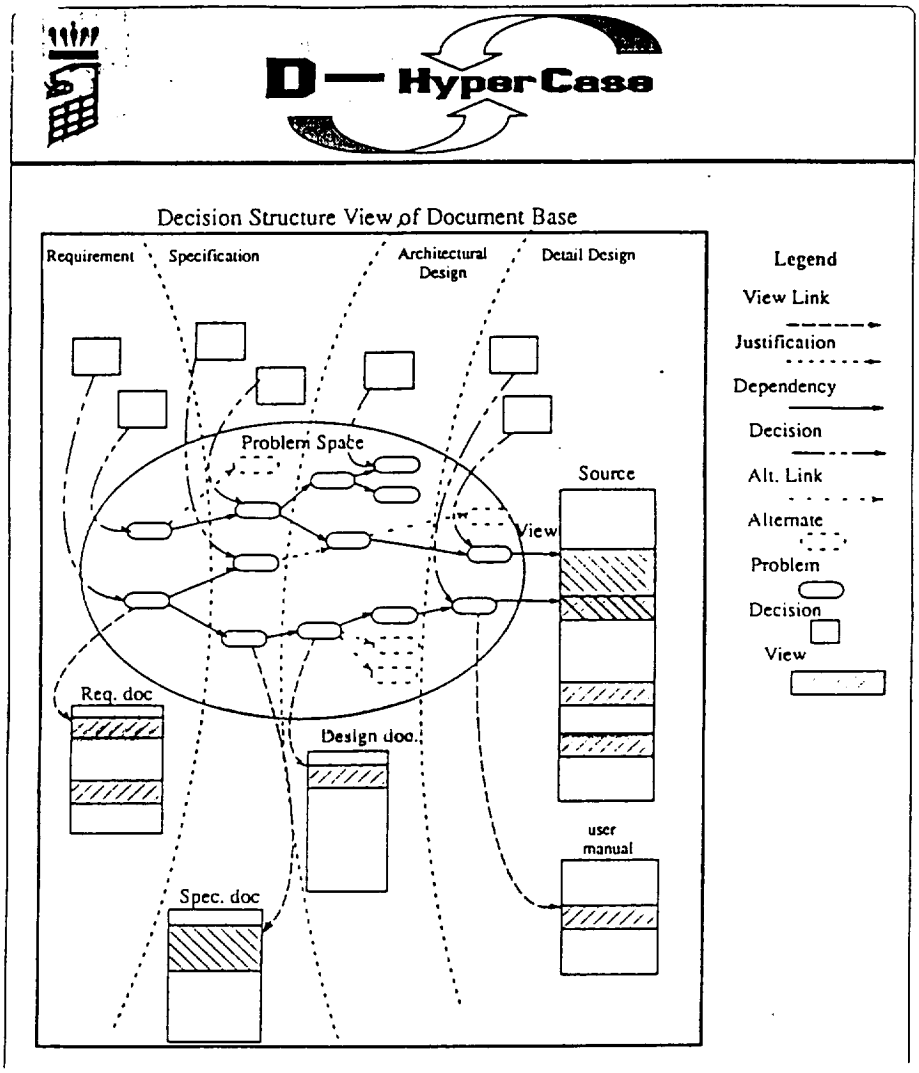


Figure 1: Decision Based Software Documentation

decisions into requirement decisions, specification decisions, etc. However, the DBSD paradigm is independent of the software development methodology used and this particular grouping is only for illustration. Since solutions to one problem may require further problem solving themselves, decision structures are linked together by a problem/solution **dependency** link. If a decision structure references or defines an artifact described in the software document base, a linkage between them is made. The decision structure provides a view of the document base which support the problem solving process. We call this view of the software document base the **decision view**.

Since design involves a careful consideration of tradeoffs among alternatives, decisions are typically made in the context of existing decisions, solutions, and policies. **Justification** links provide the context of solutions and other problems in which a decision is made. The structuring provided by decisions does not necessarily correspond to the normal **presentation structure** of a document. The Presentation Structure of a document is the organization traditionally used to present the document. This structure may be dictated by documentations standards which detail which sections must be present, their contents and relative order or by document processing programs (such as the compiler or document preparation tools). The view of a decision may be scattered throughout a document(s) impacting many parts of it. In addition, several decisions may impact the same part of the document. As shown in the figure, several decision views may overlap. Through the problem solving graph, one can trace from any document back through the relevant problems to a requirement. Or one can trace from a problem to its solution, expressed perhaps as the lines of code which implement a solution to that problem. Since requirements, specifications and design documents represent a solution at some level of abstraction, the problem solving graph provides a view of those documents. Also some of the final solutions are not programs. Users manuals and operations guides are part of the system solution and can be in the view of decisions. This figure indicates the generality of the DBSD paradigm. All phases of the life cycle can be viewed as problem solving from the generation of the initial requirements to the source code. In fact, DBSD encompasses other design activities

as well. We have applied the DBSD paradigm to the development of its process model.

In order to gain a better understanding of how DBSD paradigm can be applied to software development and maintenance, a prototype DBSD support system, called D-HYPERCASE , has been developed. D-HYPERCASE is described in [?].

3 Process Modeling

Since problems can arise at any time during development, fitting them into a problem space is not predetermined. The organization of this space and the resulting structure of the final software system is not given and will depend on how decision making is managed. For example the Spiral Model of software development [?], focuses on the high risk problems first which provides the context in which less risky problems are solved. Incremental Build delivers a system as a series of products with increasing functionality where later versions build upon decisions made during the earlier versions. We believe that DBSD is a fundamental paradigm which can be incorporated into diverse engineering methodologies. In order to better understand how to utilize DBSD we have developed a general process model for it. This process model does not prescribe the order in which decisions are made nor the set of documents produced. This process model addresses the kind of information to produce and the activities used in creating and using this information. By writing an explicit process model, the roles of different members of the software engineering team and management is clarified. In addition, it is possible to identify desirable functionality for the software engineering environment.

In developing the process model, we began to recognize the importance of managing problems which are in a state of transition. Problems which have not been fully integrated into the document base must kept be visible. A **project agenda** is used to record the state of all problems under active development. This project agenda may contain tentative alternate solutions to problems from which an assessment can be made. Once a commitment is made, the chosen solution can be linked into the document base. In recognition that software development is an on going problem solving activity, it is often possible to identify those decisions which are likely to be revisited after deployment. One of the major insights in developing the process model for DBSD is development of a active role for planned maintenance in the design decision making process. Because decision making is often hindered by lack of experience or limited ability to judge the appropriateness of a solution, many solutions will be sub-optimum or unworkable. Since in a complex system there will be many decisions which will impact the overall performance either favorably or adversely, it can be quite difficult to assign credit or blame to individual decisions. We propose that decisions supported by weak justifications be validated by instrumenting the solution to collect data which will support or refute the decision - a process we refer to as the **DIRE (Decide, Instrument, Re-Evaluate)** method of problem solving.

Example 2 *In a previous example, the response time under a worst case load was required to be under 3 seconds for 90% of the transactions. In order to validate or refute the worst case scenario, the system could be instrumented to collect watershed statistics. These statistics can help give insight into the nature and probability of worst case behavior. Subsequent decisions which affect this requirement can now be more informed.*

The process model provides for management of the development team through a series of activities. In our model the activities include

- identification of new problems.
- interaction with the software engineering environment to understand a problem and its constraining context,
- assessing different methods of solution including reuse of existing solutions
- review of proposed solutions in a decision review meeting
- update of the project document base including problems in process

- allocation of resources by management to the solution of active problems
- implementation of committed solutions

4 Experience

The original concepts of the DBSD paradigm were developed as a result of a set of experiments in performing maintenance tasks on a moderate sized production Navy application program. Since these initial experiments, we have built two Software tools supporting the DBSD paradigm. We have also developed a general process model for DBSD. To the degree it was practical all these efforts were developed using the DBSD paradigm. The following is a summary of what we learned from these experiences. More details can be found in earlier publications [?, ?, ?].

- We found the reusability often relates to solutions to problems which result in code fragments scattered through the modular structure. This particularly true if the solutions are heavily impacted by non-functional considerations.
- The time spent in understanding an existing system consumed the major portion of the effort during maintenance tasks (we measured about 80%). Furthermore, understanding the non-technical reasons why certain solutions were taken were the most difficult to reengineer.
- The first software tool was built using a graphical hypertext system developed at Old Dominion University. This system was extensively modified in adapting it to our project. The decisions structure for the original software was reverse engineered by one of the original systems designers. In order to assess the value of DBSD in this effort, a set of metrics were developed to measure the differences between the ideal and actual closures. An abstraction metric [?] measures the size of the document base associated with a viewpoint. Examples of viewpoints are decisions, modules or function points. The number of viewpoints which must be understood in order to perform a task defines a related metric called the task abstraction metric. A set of precision metrics measures the difference between the actual and ideal closures (percentage of actual closure not in the ideal and percentage of the ideal closure not in the actual). During this initial development data was manually collected to measure precision and abstraction. These preliminary results [?] indicate that by using the decision view instead of a functional view of the software, the software maintainer would be able to find the relevant parts of the document base more precisely using fewer abstractions (by a factor of 2 to 5).
- Removal of obsolete code would be easier using the decision view than a functional view of the source code document. We believe the same would hold for other forms of documentation as well.
- We do not expect that it is possible to develop a perfect decision structure. An unstated problem or assumption will of course not result in a decision structure. The penalty for changing a system with unstated decision is that they will have to reverse engineering during the understanding and impact analysis. However once articulated, there is a rapid growth in the precision of the decision structure with respect to the new and future related changes. This restructuring is much easier than that which would be required by restructuring an inappropriate modular structure for a system.
- The last observation on the ability to grow and modify the decision structure suggests that it should be possible to apply the DBSD method to existing software systems.
- The additional effort to document the decision structure was measured at an additional 10% in keystrokes entered. This reflects the decision to automatically associate the primary decision with the document as it is initially entered (that is, the software engineer, first identifies the problem they are working on, then edits the document base. All new entries are automatically associated with the current decision). We believe audio data entry would reduce the effort even further.
- The process model described in [?] is our third experience with DBSD. This effort further shows the generality of the DBSD approach. The process model deals primarily with management issues. We are currently building a third system which incorporates additional support for the process model. In particular, this system will have project agenda management and will help build and assess the closure of a maintenance task.

5 Conclusions

The Decision Based Software Development Paradigm offers a new approach to developing and maintaining complex software systems. In this approach, the process is organized around the problem solving activity rather than the product structure. Our research into the DBSD paradigm has addressed the organization of the document base, the functionality of a Software Engineering Environment and a process model to support it. It provides a different structure of the document base which is related to the process which creates and maintains it. This model supports decision validation based on the Decide, Instrument, Re-Evaluate (DIRE) paradigm in which the software system is instrumented to collect data to validate or refute decisions which are weakly justified. Our experiences with the DBSD paradigm indicate that it is sufficiently general to provide life cycle support for complex software systems.

Software Life Cycle Support: Decision Based Software Development [1]

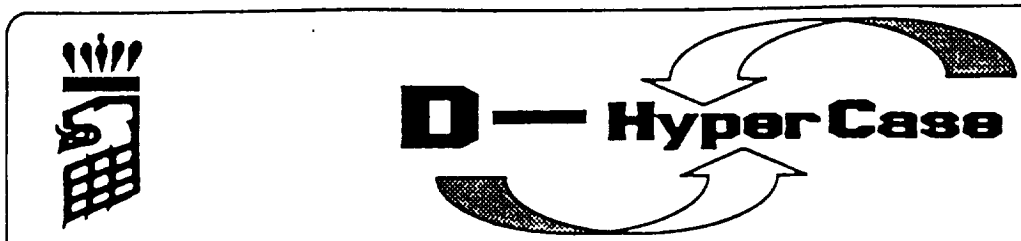
Chris Wild *Kurt Maly*
Tammy Taylor *Daniella Rosca* *Jing-Yuan Zhang*

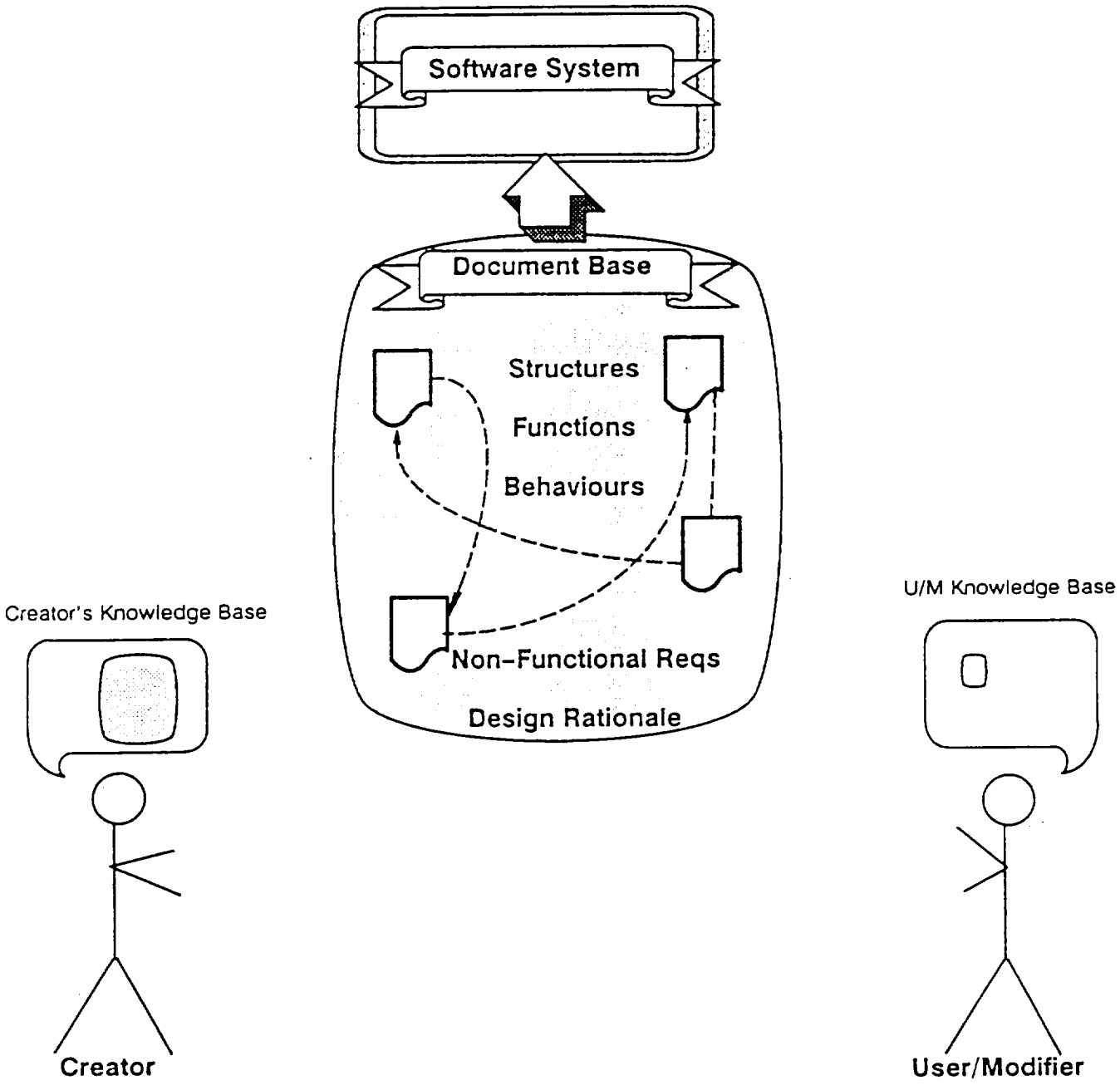
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162

[1] Work sponsored by grants NAG1-439, NAG1-966 from NASA Langley Research Center and grant CIT INF-92-008 from Virginia's Center for Innovative Technology

OUTLINE

- Background
- Decision Based Approach to Software Development and Maintenance
- D-HyperCase: Prototype Implementation
- Process Model for DBSD
- Discussion
- Conclusions


D — HyperCase


Software System
Document Base
Structures
Functions
Behaviours
Non-Functional Reqs
Design Rationale
Creator's Knowledge Base
Creator
U/M Knowledge Base
User/Modifier

What are the Problems?

- *PROBLEM:* Most Documentation Fails to Support the Process –
  Document After the Fact, Largely Irrelevent.
- *PROBLEM:* Functionally Oriented Software Engineering Methods –
  Poor Support for Non-Functional Requirements Resolution.
- *PROBLEM:* Structured Methods Oriented Around Products –
  Early Stages of Life Cycle, Policy Making, Style Concerns Ignored.
- *PROBLEM:* Rigid Structures –
  Reusability and Evolution Constrained.

Decision Based Software Development

⇒ Model the Process of Software Development and Evolution as a Set of Interrelated Problem Solving Episodes.

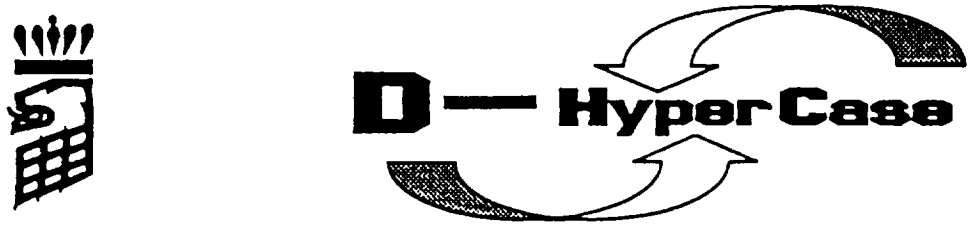
⇒ Record Decisions Made and Their Relationships to Other Decisions and to the Products of Software Development.

Why Problem Solving?

- Supports Process Across Life Cycle
- Supports Non-functional Requirements Resolution
- Process Oriented

Why Record Decisions

- Decision Rationale Difficult to Reverse Engineer
- Record Promising Alternate Solutions and Dead Ends
- Engineering is Trading Off Decisions
- Software Maintenance is Changing Decisions



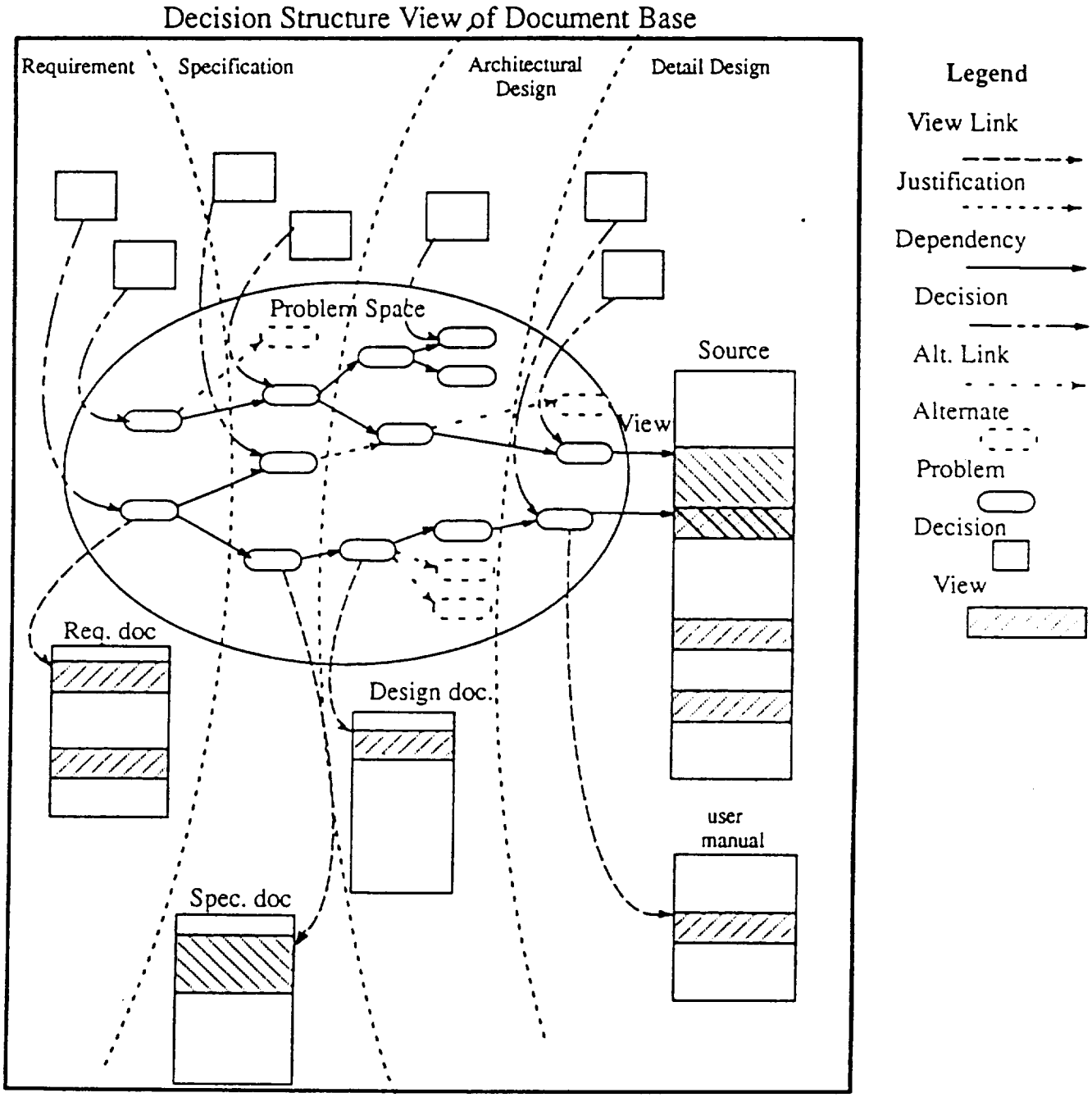
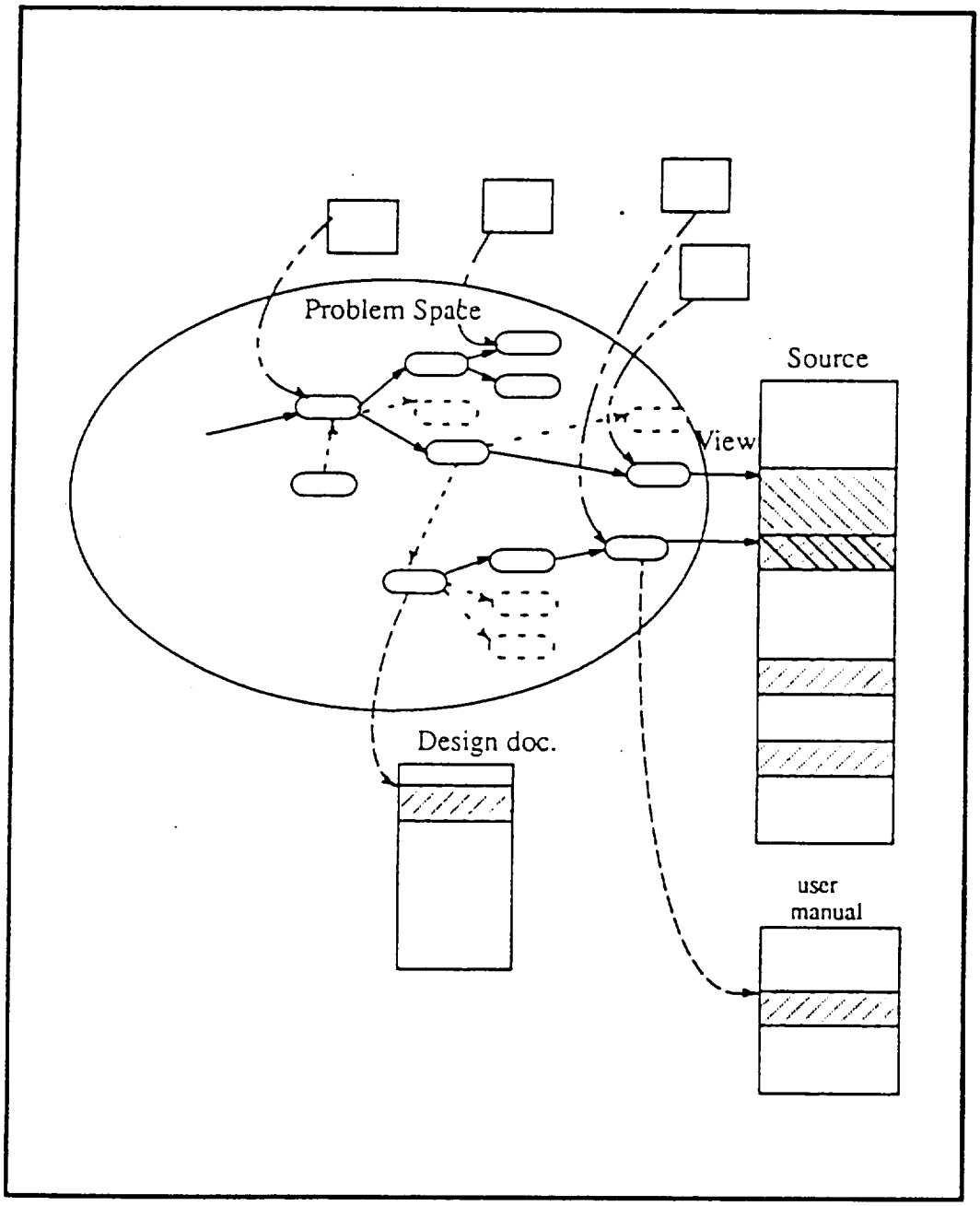


Fig. 1 Decision–Based Software Documentation



Legend

View Link

Justification

Dependency

Decision

Alt. Link

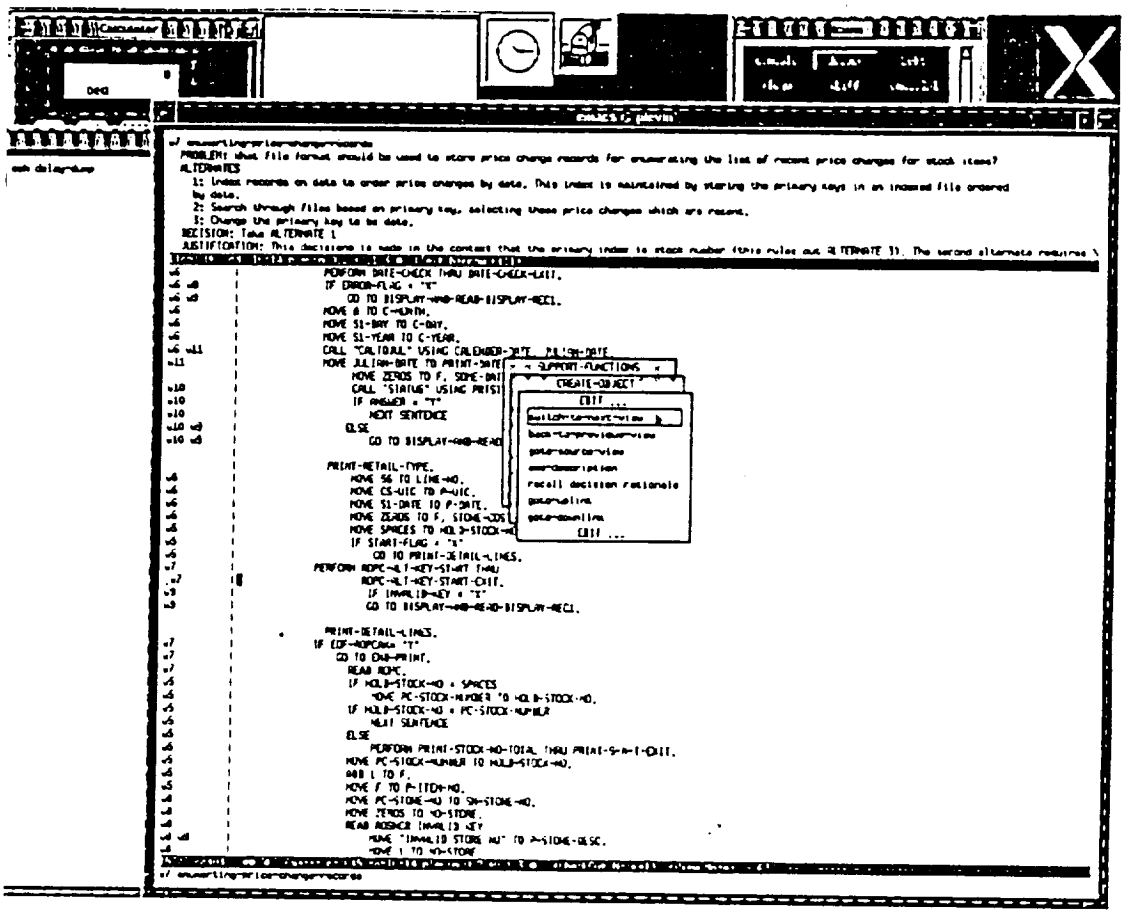
Alternate

Problem

Decision

View

Forming Closure



Software Maintenance Life Cycle

Understanding:

What problems are addressed?
How are they solved?
Why was this solution chosen?
How are the parts of the system interrelated?

Impact Analysis:

What parts of the system are affected by a decision?
What decisions impact a particular part of the system?
What Level of Effort is Required?

Redesign:

Justify and Commit Redesign
What parts of the system can be reused?
What parts are now obsolete?
Is the modification consistent with the rest of the system?

Validation:

Does the System satisfy its requirements?
If not, which decisions must be changed?

Process Modeling

Process:

- Set of activities
- Relationships among Activities
- Team Structure and Responsibilities
- Role of Software Engineering Environment
- Policy and Constraints
- Resource Management

Process Model: Formalization of a design method which

- Provides notations amenable to analysis
- Encompasses breadth and depth of software development
- Assists in management - defines milestones
- Provides heuristics/algorithms for well-understood situations

Process Model for Traditional Life Cycle

Detailed Description

External_customer_requirement_resolution$_{CU}$ --> *task*$_{CU}$ > (Software_development | Customer_feedback) > system

Internal_perfection_and_correction$_{CU,MA}$ --> *task*$_{CU,MA}$ > Software_development > system

Software_development --> task > (*Understanding*$_{MA}$
Understanding$_{SE}$) >/*requirements_definition*/ task_root: task_problem
/*list_of_reusables candidates*/ {problem},
Task_problem_solving > /*first_level_decomposition*/ {task_problem},

{*Assessment*$_{MA,SE}$ > (task_root, effort, task_root.size, task_root.risk) Change_task_decision > task_root}

Assign_resources > schedule

Transfer_task_to_problem_space > agenda

{agenda; schedule > *Solve_problem*$_{SE}$ > agenda
Review_meeting > meeting_notes
Implement_meeting_decision > schedule, agenda}

Understanding$_{MA,SE}$ --> Exploring || *add_to_report*$_{MA,SE}$ > report

Exploring --> Requirements_definition || Reusability_search

Requirements.definition --> (keywords > **locate_problem** > relevant_nodes: {problem},
make_new_requirement) > task_root_problem

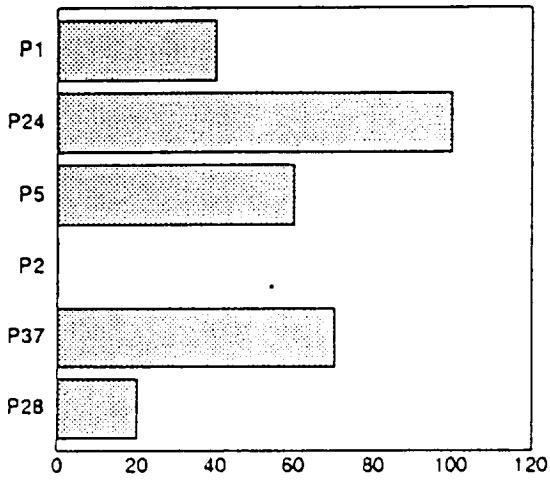
∇ relevant_nodes > Understand_problem

Understand_problem --> problem >{(Visit_node **dependency_up** > problem)
| **terminate_at_node_closure_relevant_nodes**
| **back**}

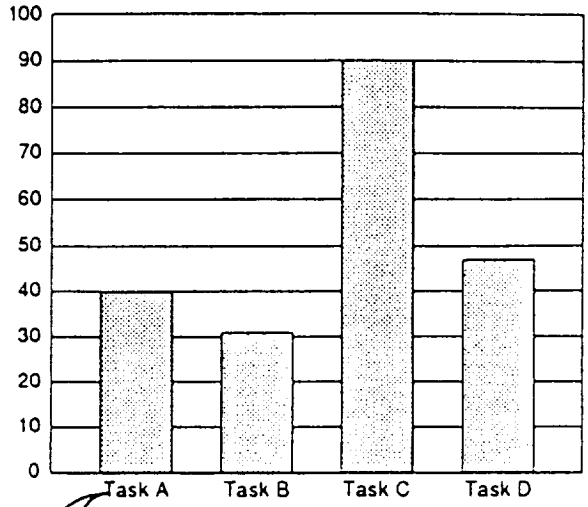
/*exploration*/ (**justification_from** > problem
| **justification_to** > problem
| **dependency_up** > problem
| **dependency_down** > problem)

D — HyperCase

Problem Status for Task A



Task Status for Release 12.4



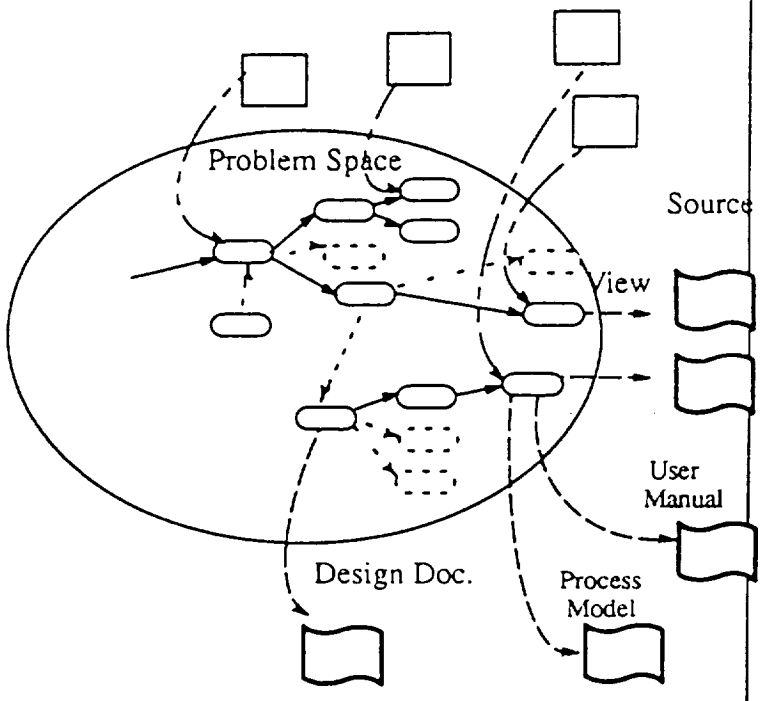
Problem Sp for Task: Reduce Response Time (Task A)

Status of Problem P37

	#	*Size*	*Min*	*Max*
New				
Del				
Reuse				
Open				
Total				

Open Problems Unassessed

P45
P135
P42
P56



Managing Deferred Decisions

- Problem Not Adequately Understood
- Solution Identified First
- Inadequate Justification
- Instantiating a More General Problem
- Inadequate Resources
- Unexpected Opportunities
- Inexperience

DIRE - Decide, Instrument, Re-Evaluate

Problem

- Engineering Tradeoffs involve many interacting decisions.
- Inexperience affects quality of some decisions.
- Impact on overall performance of decisions may be difficult to determine.

Solution

- Identify decisions with "weak" justifications.
- Minimize the impact of these decisions (minimize justification links).
- Instrument system to validate or refute this decision.
- Re-evaluate decision using data collected from operational system.

Evaluating the Process Model

- Assess degree to which the objectives have been met
- Develop a better understanding of the dynamics of the process
- Identify what pieces of information are needed and when
- Develop a methodology for DBSD
- Identify where tools could be applied to improve the process

Interval of Evaluation

Main purpose of evaluation is self improvement.
Balance **progressive** and **anti-regressive** activities.
Four intervals of evaluation:

System Life Time: A cost/benefit analysis is done over the life time of the entire system.

Release: Most large software systems go through a set of releases over its life time. Each release usually represents a significant change in the system in which major problems are rectified and new features are added.

Task/Change Order: A task represents a unit of work to be performed. A task could be defined in response to a trouble report or could represent the addition of some new feature.

Session: A session represents a contiguous period of time during which the programmer is working in the development environment.

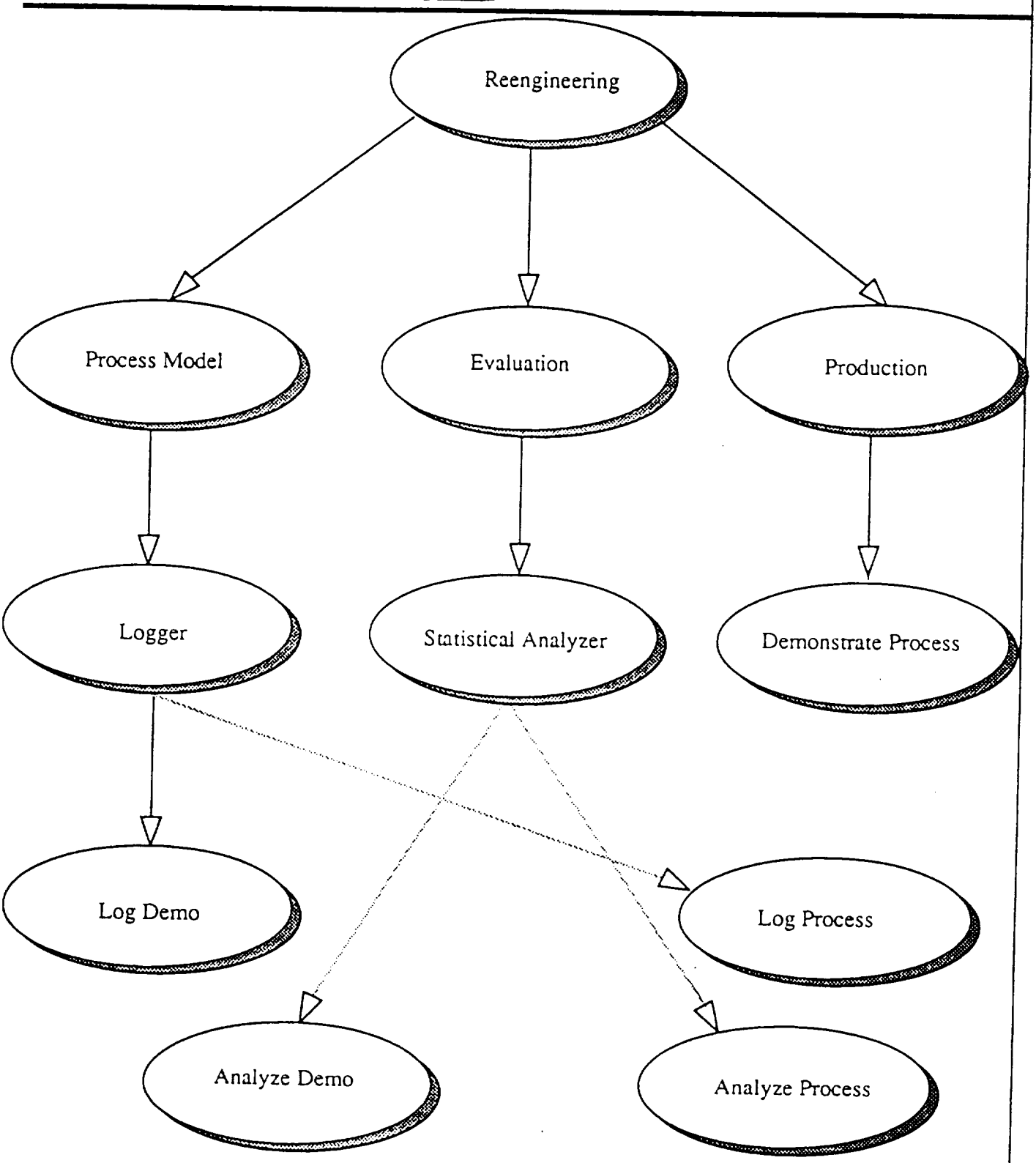
Understanding the Dynamics of the Process

- Is there a working set model for software maintenance tasks?
- How are the dynamics of the process affected by the task type (corrective, perfective, adaptive)?
- What information was found useful in performing a task?
- What are the different information needs of Managers, Software Engineers, Others?
- What information was missing or difficult to access?
- How can consistency be maintained?

Session/Tasks Measures gathered:

- Sequence of decisions visited
- Sequence of commands issued
- Time spent in each view
- Keystrokes entered

D — HyperCase


Reengineering
Process Model
Evaluation
Production
Logger
Statistical Analyzer
Demonstrate Process
Log Demo
Log Process
Analyze Demo
Analyze Process

Summary of Recent Experiments

Data Collected over 43 Sessions Resolving 10 Maintenance Tasks.

- Additional Documentation Effort for the Decision Structure is about 10%.
- About 80% of Time was Spent in Reading This Documentation.
- About 90% Of The Accesses To The Document Base Were Through The Problem Views.
- The failure rate of reused code was 5 times less than new code.
- Changing decision on cursor position in second window involved the deletion of one problem whose view contained 315 LOCs in 25 functions (containing a total of 892 LOCs).
- Corrective Maintenance tasks are more localized with high holding times.
- Perfective Maintenance tasks access many views with high holding times.
- Adaptive Tasks access a few views with short holding times then create new views with high holding times.

Future Directions and Work In Progress

- Create Version 2 DHC - Agenda and Task Management Support
- Prepare Guide Book Based on Process Model/DHC
- Undertake Empirical Evaluation of Process Model
- Study Support for Reusability
- Active Environmental Support for Process Model

Conclusions

- To perform a given task, the decision viewpoint requires fewer conceptualizations than the functional viewpoint.
- These conceptualizations are more precise in identifying the relevant parts of the document base for revision.
- DBSD helps control the sometimes chaotic creative process of software development.
- Non-functional requirements play an important role in the problem solving process
- Explicit Process Model clarifies roles for Software Engineers, Managers, Operators and Software Tools.
- Software Engineering Environment should support Process Model.
- Software System should be instrumented to collect information to validate/refute critical decisions

D — HyperCase

Library of Components

C1

C2

. . . .

Cn

Compositional

Set of Decisions

V1

V2

V3

. . . .

Vm

Generative

Set of Decisions
&
Process of
Generation

V1

V3

. . . .

Vk

Models of Reuse

Closure

Ideal closure: exactly those portions of the document base which are relevant to the task.

Actual closure: depends on the structuring method and the granularity of view point it imposes on the document base.

Precision:

The degree of match between the actual and ideal closures.

Relevance: The percentage of the actual closure which is in the ideal closure. One minus the density is a measure of the "noise" in the actual closure.

Oversight: The percentage of the ideal closure which is not in the actual.

Abstraction

The representation of a set of related concepts as a single unit. If this representation helps in the understanding of the set of concepts then it is an **appropriate** abstraction.

Size: One measure of an abstraction is the size of the problem it conceptualizes. The metric used in our evaluation is the number of Lines Of Code (LOC) related to the abstraction.
More LOC ⇒ More Abstract.

Power: For a given task, the number of abstractions needed to understand and solve that task is a measure of the power of the set of abstractions.
Smaller Sets ⇒ More Power.

Using DBSD Approach

- Make the Solution Visible.
  Work Through a Particular Task.
- Identify the Problems That Each Solution Solves.
- Generalize the Problem, if Appropriate.
- Uncover the Underlying Assumptions Which Justify the Particular Solution.
- Develop Alternate Solutions.
- Justify the Assumptions. Pick the "Best" Solution.
- Link the Identified Problems and Justifications.
- Place Incomplete Decisions on Project Agenda

Cost - dwindling budget - increased demand on increasing productivity and reliability. - competiveness.
loss of intellectual control of process. Points:

- paradigm justification
- DBSD
- DHC
- Evaluation purpose
- results
- methodology and process

– management controls temporal aspects, resources and defines non-technical constraints - interleave VG – management is process of making decisions - dynamics of decision making - solution w/o problem, SP wo alternates, alternates wo solution, no justifications – degree of satisfaction If time show them performance req. and DO without solution and justifications are hypothetical.

1. do nothing
2. change req.
3. temporary release from req.
4. tune program
5. redesign

–DO create interleave for this set of decisions. No justifications for choosing yet. limited resources could push for 2 or 3. TO choose between 2 and last two need to know if it scales (complexity constant, linear, exp.). Points up that alternates are part of the agenda.

Decision Based Software Development in the Reuse Arena

Chris Wild *Kurt Maly*
Daniela Rosca *Chenglin Zhang* *Tammy Taylor* *Chris Cowles*

Department of Computer Science
Old Dominion University
Norfolk, VA 23529–0162

Work sponsored by grants 126581 from NASA Langley Research Center, grant 526291 from Virginia's Center for Innovative Technology and grant 526292 from Paramax

OUTLINE

- Software reuse terminology
- Decision Based Software Development Approach
- Decision Based Software Development in the Reuse Arena

ABSTRACTION

What type of software entities are reused and what abstractions are used to describe them?

Reusable code:

- reusable data–centered components (Ada: stacks,lists, strings, queues,sets, trees,etc);
- reusable function–centered components(sorting, searching).

Software Schemas (formal extension to reusable software components) :

- the emphasis is on reusing abstract algorithms and data structures
- each schema has a specification that includes:
  - a formal semantic description of the schema
  - assertions for correctly instantiating the schema (constraints on the variable part of the schema)
  - assertions for the valid use of instantiated schema (preconditions, postconditions)

Application Generators:

- reuse complete software system designs (expert systems generators, compilers generators, etc.)

Very High–Level Languages:

- can be viewed as specification languages when compared with high–level languages (executable specification languages)
- mathematical abstractions (set theory, constraint equations)

Transformational systems:

- development histories that can be replayed (PADDLE, Glitter)
- transformations : mappings from syntactic patterns of code into functionally equivalent, more efficient patterns of code.

Software Architectures:

- large grain software frameworks and subsystems that capture the global structure of a system design

SELECTION

How are reusable entities selected for reuse?

Reusable code:

- techniques for describing the components (formal annotations – Anna)
- techniques for classification and retrieval (different indexes)

Software Schemas:

- sophisticated searching at the abstraction specification level (PARIS)

Application Generators:

- libraries of application generators have not been developed yet

Very High–Level Languages(VHLL):

- selecting the VHLL that is most appropriate for a particular application
- selecting the language constructs that best represent the application

Transformational systems:

- expert systems technology to select from a library of transformations (Glitter)

Software Architectures:

- library techniques

SPECIALIZATION

How are generalized entities specialized for reuse?

Reusable code:

- by directly editing the code
- parameterized macro expansions
- Ada generics
- inheritance

Software Schemas:

- substitution of language constructs, code fragments, specifications, or nested schemas
- choosing from a predefined enumeration of options

Application Generators:

- by providing an input specification. The techniques used depend on the application domain abstractions: grammars, regular expressions, graphical languages, interactive dialog,etc.

Very High–Level Languages:

- parameterized language constructs that are specialized by recursively substituting other language constructs

Transformational systems:

- not an issue, typically

Software Architectures:

- horizontal: source–to–source transformations (optimizations in Draco)
- vertical: component refinements (alternative implementations with different performance characteristics)

INTEGRATION

How are reusable entities integrated to create a complete software system?

Reusable code:

- module interconnection languages

Software Schemas:

- module interconnection languages
- semantic specifications to compose schemas:
  - horizontal composition corresponds to schema composition using nested schemas
  - in vertical composition higher–levels of abstractions are created.

Application Generators:

- do not require integration

Very High–Level Languages:

- encapsulated computation (computation within a function is influenced only by its input parameters and return values from the functions it calls)– PAISLey
- order–independent specification and compiler–generated control flow and data flow – MODEL

Transformational systems:

- implicit in the order in which the transformations are applied

Software Architectures:

- special purpose module interconnection language

Decision Based Software Development

⇒ Model the Process of Software Development and Evolution as a Set of Interrelated Problem Solving Episodes.

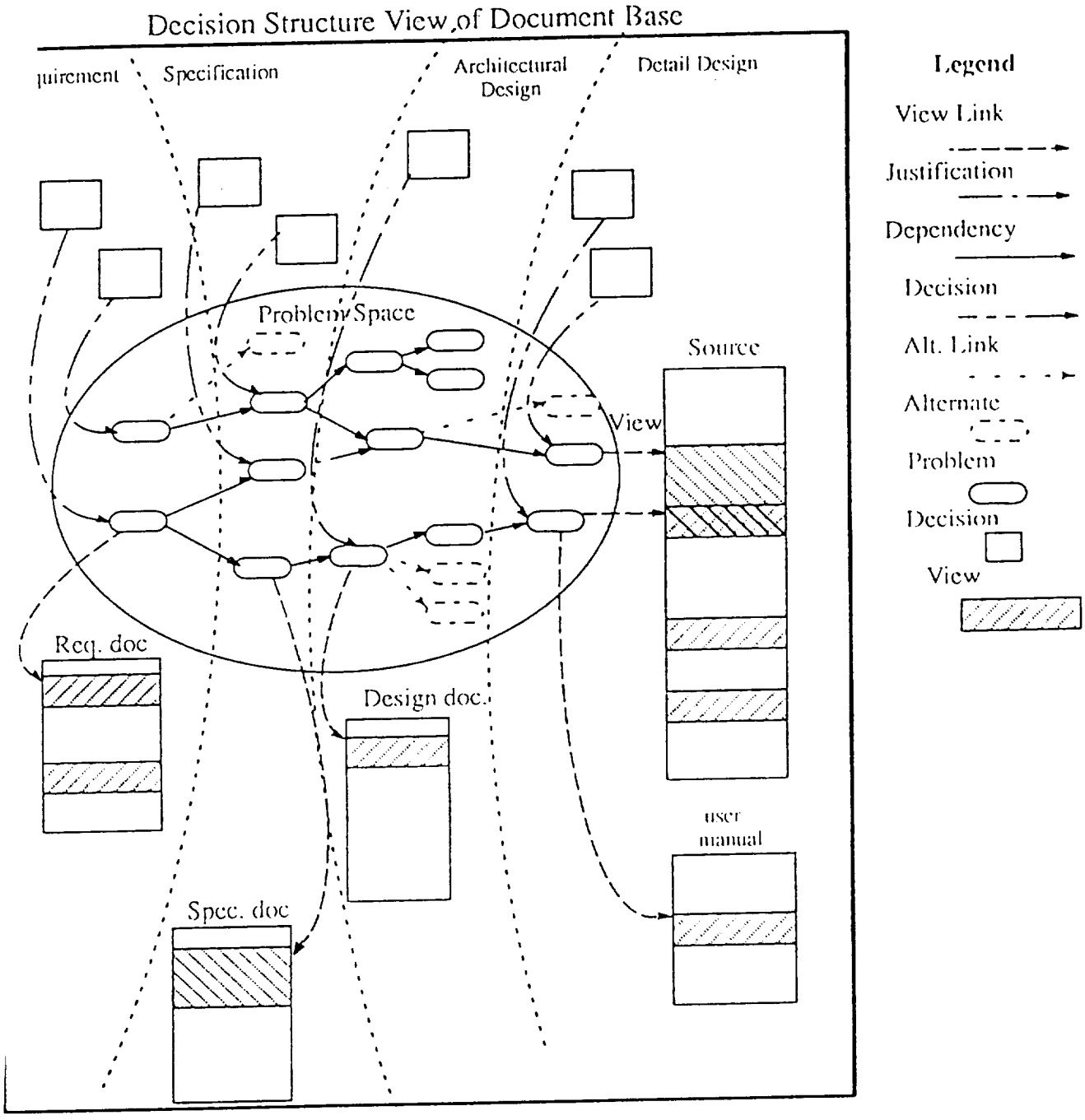
⇒ Record Decisions Made and Their Relationships to Other Decisions and to the Products of Software Development.

Why Problem Solving?

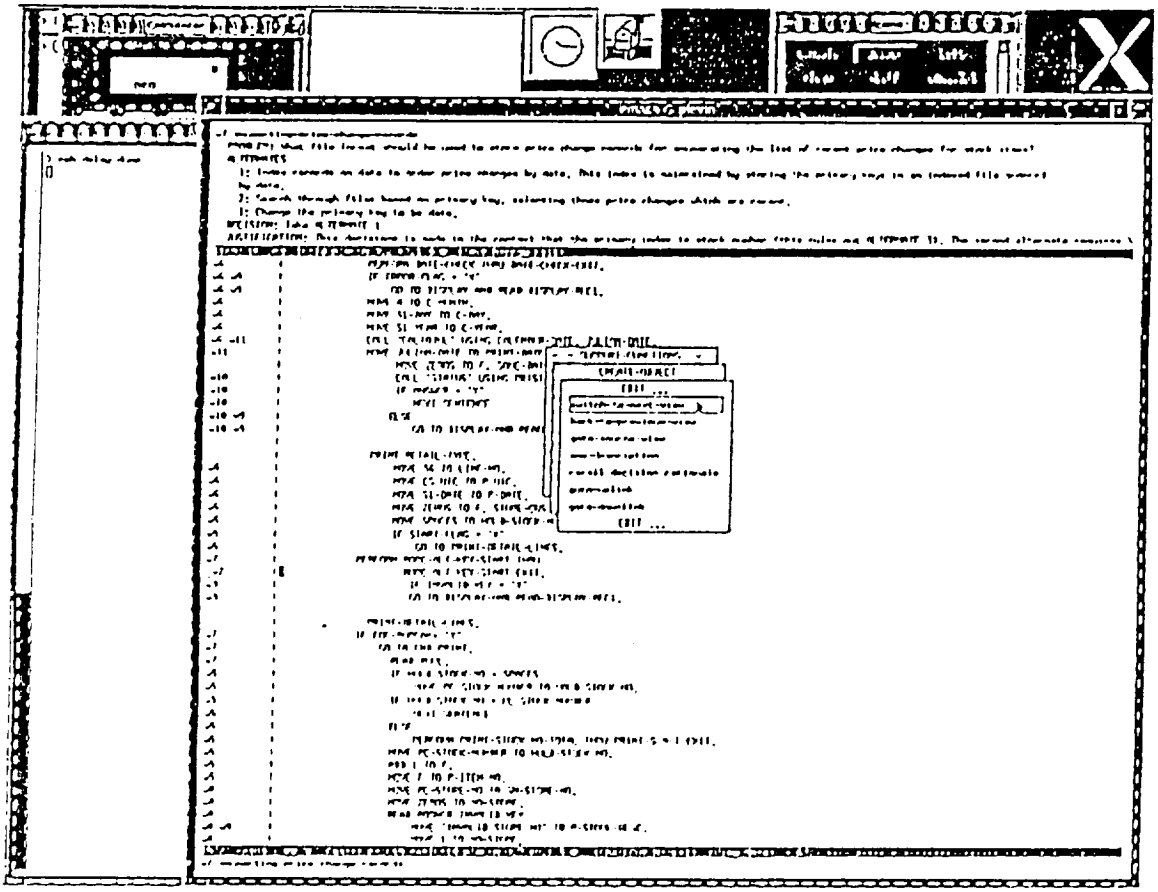
- Supports Process Across Life Cycle
- Supports Non-functional Requirements Resolution
- Process Oriented

Why Record Decisions

- Decision Rationale Difficult to Reverse Engineer
- Record Promising Alternate Solutions and Dead Ends
- Engineering is Trading Off Decisions
- Software Maintenance is Changing Decisions



Decision–Based Software Documentation



ORIGINAL PAGE IS
OF POOR QUALITY

Software Maintenance Life Cycle

Understanding:

What problems are addressed?
How are they solved?
Why was this solution chosen?
How are the parts of the system interrelated?

Impact Analysis:

What parts of the system are affected by a decision?
What decisions impact a particular part of the system?
What Level of Effort is Required?

Redesign:

Justify and Commit Redesign
What parts of the system can be reused?
What parts are now obsolete?
Is the modification consistent with the rest of the system?

Validation:

Does the System satisfy its requirements?
If not, which decisions must be changed?

Process Modeling

Process:

- Set of activities
- Relationships among Activities
- Team Structure and Responsibilities
- Role of Software Engineering Environment
- Policy and Constraints
- Resource Management

Process Model: Formalization of a design method which

- Provides notations amenable to analysis
- Encompasses breadth and depth of software development
- Assists in management - defines milestones
- Provides heuristics/algorithms for well-understood situations

Summary of Recent Experiments

Data Collected over 43 Sessions Resolving 10 Maintenance Tasks.

- Additional Documentation Effort for the Decision Structure is about 10%.
- About 80% of Time was Spent in Reading This Documentation.
- About 90% Of The Accesses To The Document Base Were Through The Problem Views.
- The failure rate of reused code was 5 times less than new code.
- Changing decision on cursor position in second window involved the deletion of one problem whose view contained 315 LOCs in 25 functions (containing a total of 892 LOCs).
- Corrective Maintenance tasks are more localized with high holding times.
- Perfective Maintenance tasks access many views with high holding times.
- Adaptive Tasks access a few views with short holding times then create new views with high holding times.

Closure

Ideal closure: exactly those portions of the document base which are relevant to the task.

Actual closure: depends on the structuring method and the granularity of view point it imposes on the document base.

Precision:

The degree of match between the actual and ideal closures.

Relevance: The percentage of the actual closure which is in the ideal closure. One minus the density is a measure of the "noise" in the actual closure.

Oversight: The percentage of the ideal closure which is not in the actual.

Abstraction

The representation of a set of related concepts as a single unit. If this representation helps in the understanding of the set of concepts then it is an **appropriate** abstraction.

Size: One measure of an abstraction is the size of the problem it conceptualizes. The metric used in our evaluation is the number of Lines Of Code (LOC) related to the abstraction.
More LOC $\Rightarrow$ More Abstract.

Power: For a given task, the number of abstractions needed to understand and solve that task is a measure of the power of the set of abstractions.
Smaller Sets $\Rightarrow$ More Power.

DBSD IN THE REUSE ARENA

Abstraction: Which entities do we manipulate and store?

- high level objects(specifications,requirements, designs) and source code (see page 31)
- in DBSD these objects are represented as: problems, alternatives, evaluations, decisions, justifications (see pages 17,18,19)
- the granularity of the objects reused is not influenced by the syntactic constructs of the source language used, but by the abstraction level of decisions made.
- DBSD allows not only the reuse of completed products, but also to reuse or replay the process used to obtain the product.
- DBSD is not centered on functional decomposition or other traditional methods, it provides a complementary approach by its non–linear view of the problems.

Selection: How do we best identify the components most relevant to a given user's needs?

What kind of taxonomies and search strategies do we provide?

- Library techniques: faceted representation, search with a similarity measure
- graphical browsing (see pages 20,21,22)
- hypertext (see pages 23,25,26,27)

FUTURE DEVELOPMENTS

- **Specialization: How do we customize a selected generic entity?**
  - selecting a value from a precomputed list of alternatives
  - transforming an abstract program schema using a set of transformational rules
  - inferring a value using a set of heuristic design rules and/or algorithms.

- **Integration: What techniques of program composition do we use?**
  - domain–specific rules
  - expert systems technology

- **Management: assessment of reuse**
  - associating views to source and documents we can exactly identify those components relevant to a solution .
  - we can identify all the components impacted by a view and estimate their size, complexity, efforts,etc.(see page 32)
  - from a specific retrieved component we can select only the relevant parts to the current problem.

DHC OUTPUT IN LATEX

PROBLEM: reengineering_problem
Develop a methodology and supporting tools to port applications from a source machine to another target machine, to translate from language A to language B and to enhance existing applications.
ALTERNATIVES: 1) Develop a fully automated, non-interactive system for specific cases. (for example, an Expert System for transforming a COBOL program) 2) Develop an expert system using only DHC. 3) Develop an interactive system using both DHC and Refinery. 4) Non-automated system for a specific case.(like "awk")
DECISION: 3) Develop a Reengineering_process_model, a running, effective DHC_prototype, make Refinery part of the environment, and link DHC and Refinery into a Unified_environment.
JUSTIFICATION: Dictated by : a. the availability of DHC and Refinery; it will be a matter of evaluation (see Evaluation_problem) b. validation of the above decision
EVALUATION:
UPLINKS: apply_and_improve_DBSD_methodology
DOWNLINKS: porting_problem translating_problem enhancing_problem

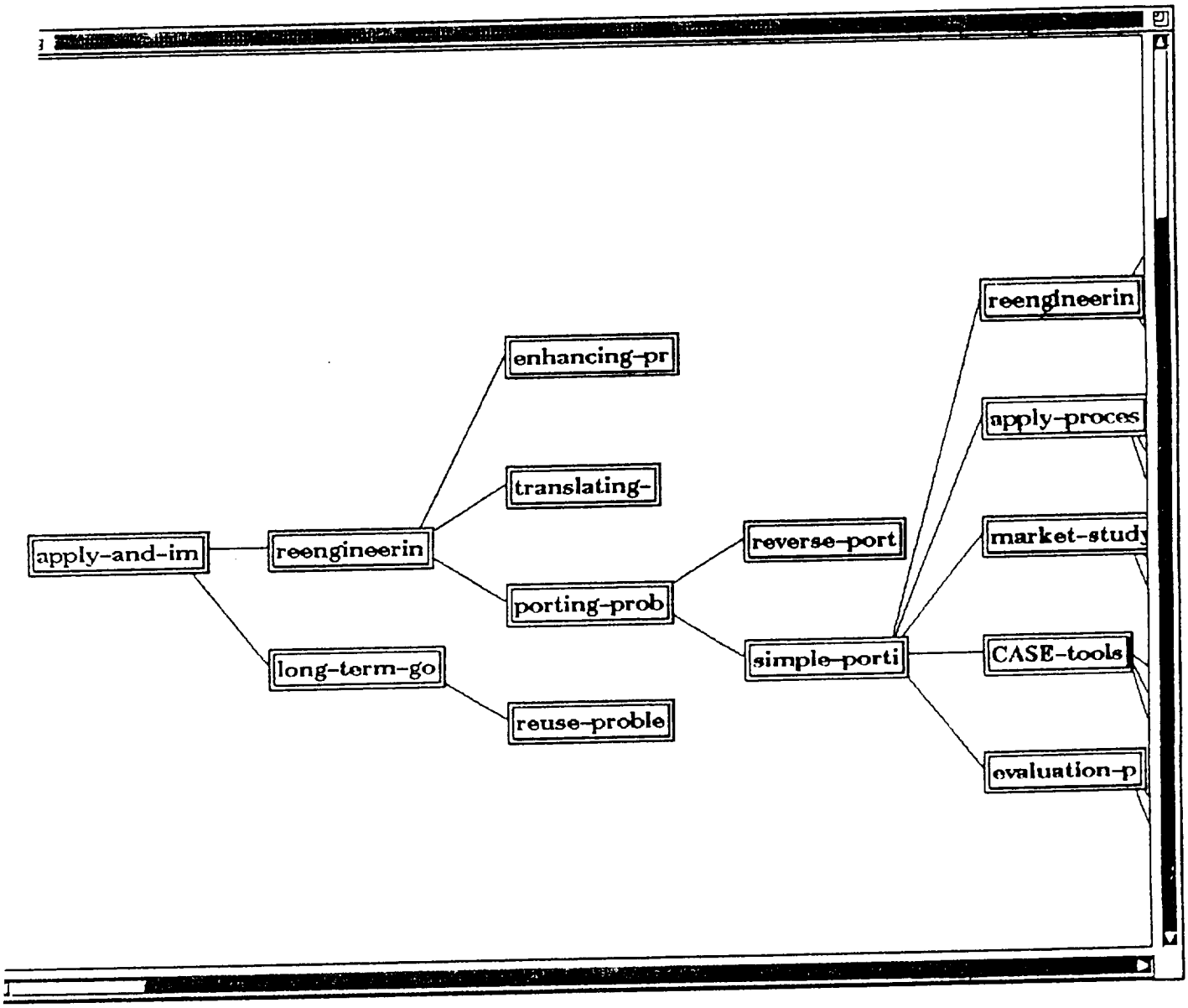
PROBLEM: reengineering_process_model
Develop a process model for porting,translating and enhancing applications
ALTERNATIVES: 1) Develop separate methodologies for: a. Porting to different dialects of COBOL(e.g. Honeywell to Microfocus) b. Enhancing applications (e.g. using SQL language instead of file operations) c. Translating into another language (e.g. COBOL to Ada) 2) Develop a general methodology for everything.
DECISION: 1)
JUSTIFICATION: The process is too little understood to fully develop a general methodology for everything.
EVALUATION: We assume that the existing application is partially DHC-ed.
UPLINKS: simple_porting
DOWNLINKS: process_model_implementation process_model_notations get_Refinery_process_model get DHC_process_model_for_Rengineering combine_DHCRfinery_process_models

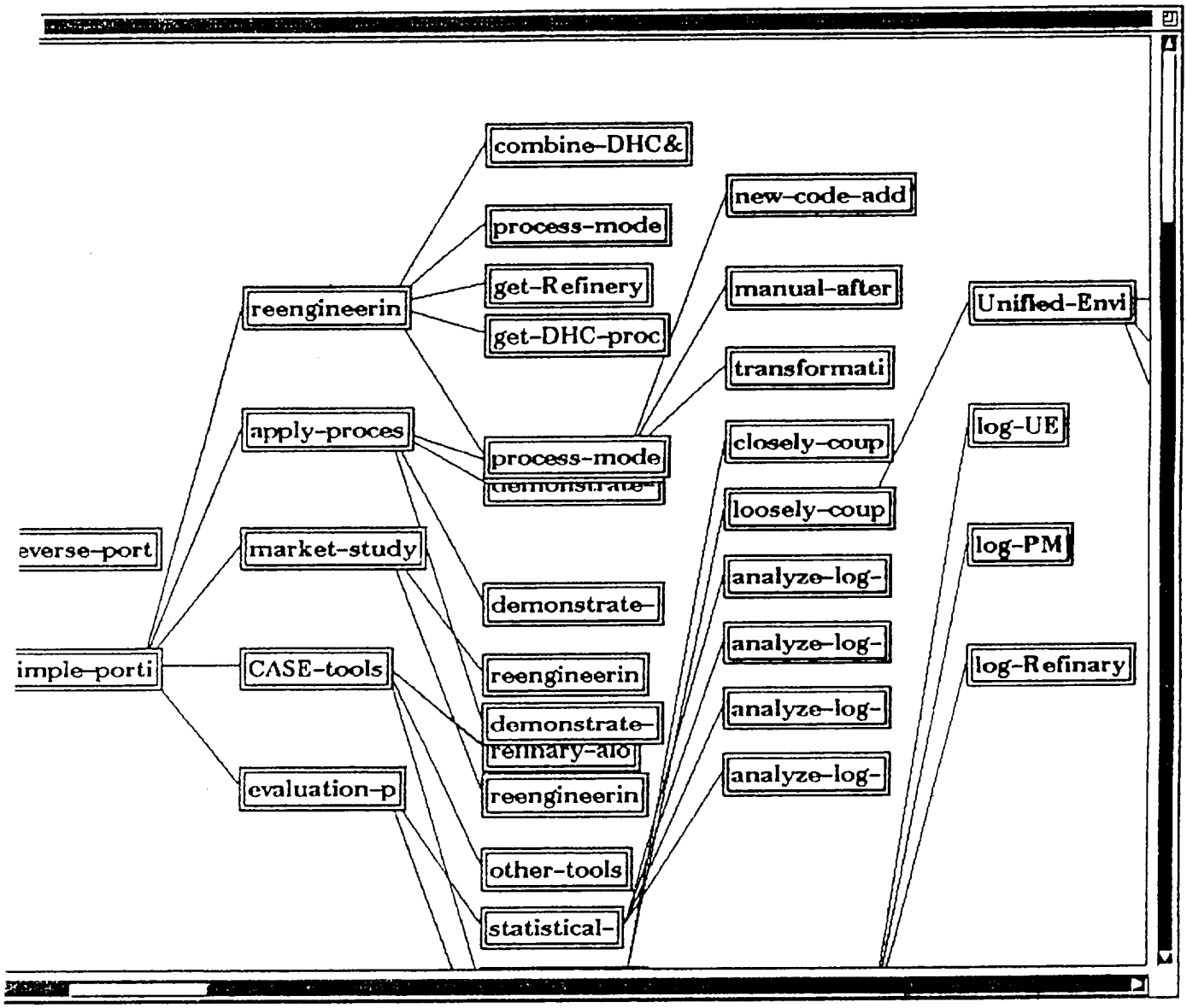
DHC OUTPUT IN LATEX

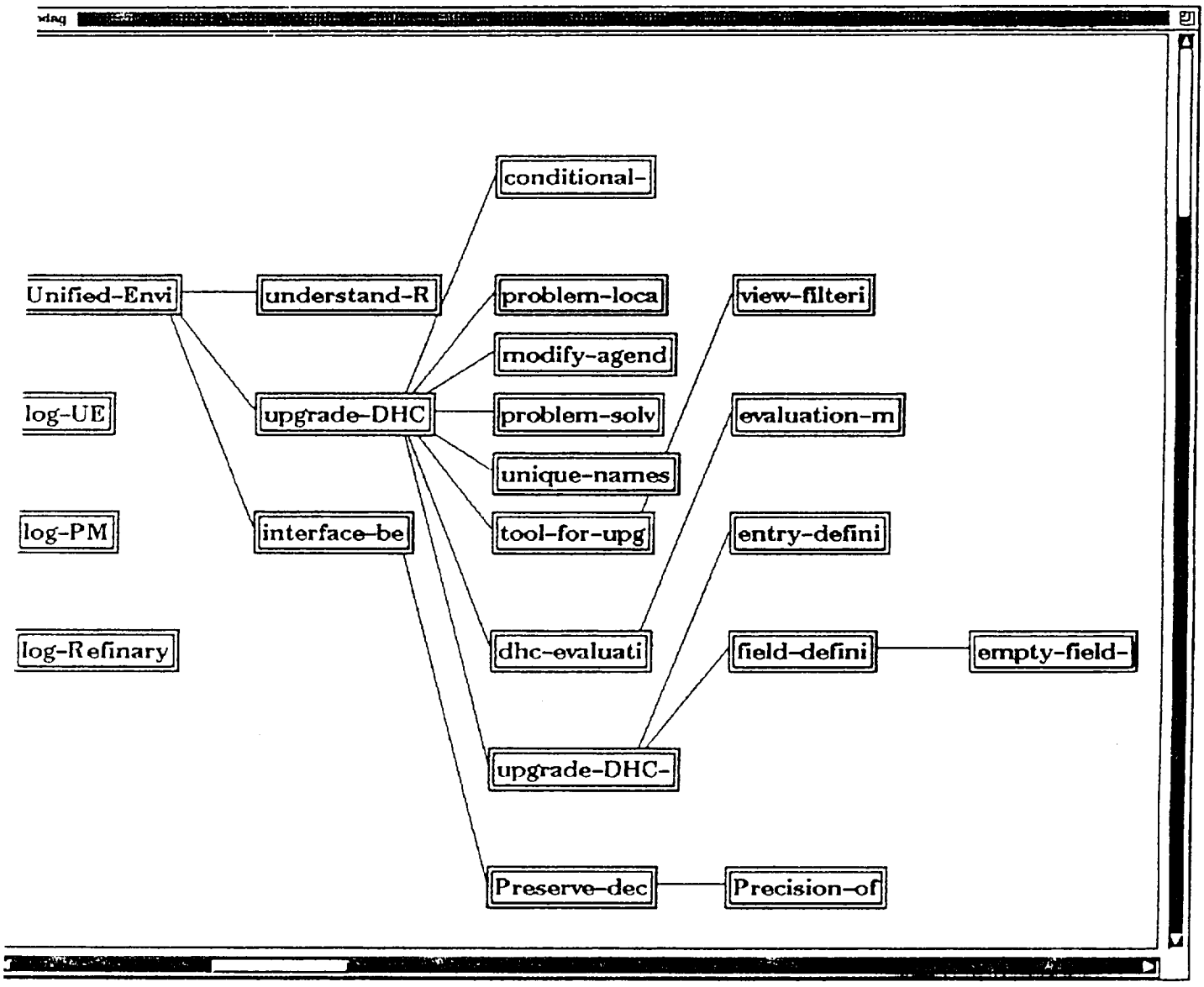
PROBLEM: reengineering-problem
Develop a methodology and supporting tools to port applications from a source machine to another target machine, to translate from language L1 to language L2 and to enhance existing applications.
ALTERNATIVES: 1) Develop a fully automated, non-interactive system for specific cases. (for example, an Expert System for transforming a COBOL program) 2) Develop an expert system using only DHC. 3) Develop an interactive system using both DHC and Refinery. 4) Non-automated system for a specific case.(like "awk")
DECISION: 3) Develop a Reengineering-process-model, a running, effective DHC-prototype, make Refinery part of the environment, and link DHC and Refinery into a Unified-environment.
JUSTIFICATION: Dictated by : a. the availability of DHC and Refinery; it will be a matter of evaluation (see Evaluation-problem) b. validation of the above decision
EVALUATION:
UPLINKS: apply-and-improve-DBSD-methodology
DOWNLINKS: porting-problem translating-problem enhancing-problem

DHC OUTPUT IN LATEX

PROBLEM: reengineering-process-model
Develop a process model for porting,translating and enhancing applications
ALTERNATIVES: 1) Develop separate methodologies for: a. Porting to different dialects of COBOL(e.g. Honeywell to Microfocus) b. Enhancing applications (e.g. using SQL language instead of file operations) c. Translating into another language (e.g. COBOL to Ada) 2) Develop a general methodology for everything.
DECISION: 1)
JUSTIFICATION: The process is too little understood to fully develop a general methodology for everything.
EVALUATION: We assume that the existing application is partially DHC-ed.
UPLINKS: simple-porting
DOWNLINKS: process-model-implementation process-model-notations get-Refinery-process-model get-DHC-process-model-for-Rengineering combine-DHC-Rfinery-process-models


apply-and-im
reengineerin
enhancing-pr
translating-
porting-prob
long-term-go
reuse-proble
reverse-port
simple-porti
reengineerin
apply-proces
market-study
CASE-tools
evaluation-p


combine-DHC&
process-mode
get-Refinery
get-DHC-proc
reengineerin
apply-proces
process-mode
everse-port
market-study
demonstrate-
imple-porti
CASE-tools
reengineerin
demonstrate-
evaluation-p
reengineerin
other-tools
statistical-
new-code-add
manual-after
transformati
closely-coup
loosely-coup
analyze-log-
analyze-log-
analyze-log-
analyze-log-
Unified-Envi
log-UE
log-PM
log-Refinary


conditional-
Unified-Envi
understand-R
problem-loca
view-filteri
modify-agend
log-UE
upgrade-DHC
problem-solv
evaluation-m
unique-names
log-PM
interface-be
tool-for-upg
entry-defini
log-Refinary
dhc-evaluati
field-defini
empty-field-
upgrade-DHC-
Preserve-dec
Precision-of

emacs: Emacs @ wurttemberg

```
                Process Model for Traditional Life Cycle

                        Kurt Maly and Chris Wild

                              Short Form


-------------------------------------------------------------------------------
. BNF PRODUCTIONS
-------------------------------------------------------------------------------

. Customer_requirement_resolution--> Software_development || Customer_feedback

. Internal_development --> Software_development

. Software_development --> (Understanding_needs  MA,SE
Modify_or_create_new_task  SE
Calculate_effect_and_costs  MA,SE
Assign_resources_to_schedule  MA
Transfer_task_to_problem_space  SE)
|| (Review_meeting_reports_and_progress  MA,SE
Delete_decision_and_replace  MA,SE
Implement_meeting_decision  MA,SE)

4. Understanding_needs --> Exploring_needs || add_to_report  MA,SE

5. Exploring_needs --> Requirements_definition Create_and_add_new_task_problem \
Find_and_add_reusable_node

6. Requirements_definition --> (locate_problem | make_new_requirement)
 Understand_problem_links

7. Understand_problem_links --> {(Visit_and_read_node dependency_up)
 | terminate_at_node_closure_relevant_nodes}
 | ( justification_from
 | justification_to
 | dependency_up )


8. Visit_and_read_node --> read_description document_view Read_document Read_ju\
stification

9. Read_document --> {(switch_view | scroll_view | emacs_commands
--%%-Emacs: dhc-process.idx  (Fundamental)----Top----------------------------
u:uplink, d:downlink, c:current, l:latex, g:graph, f:form, i:index, p:process,
```

emacs: Emacs @ wurttemberg

```
ROBLEM: reengineering-problem
Develop a methodology and supporting tools to port applications from a source \
achine to another target machine, to translate from language L1 to language L2\
and to enhance existing applications.
LTERNATIVES:  1) Develop a fully automated, non-interactive system for specifi\
 cases.
for example, an Expert System for transforming a COBOL program)
     2) Develop an expert system using only DHC.
     3) Develop an interactive system using both DHC and Refinery.
     4) Non-automated system for a specific case.(like "awk")
ECISION:  3) Develop a Reengineering-process-model, a running, effective
HC-prototype, make Refinery part of the environment, and link DHC and
efinery into a Unified-environment.
USTIFICATION:  Dictated by :
        a. the availability of DHC and Refinery; it will be a matter of evaluat\
on (see Evaluation-problem)
        b. validation of the above decision
VALUATION:

PLINKS:  apply-and-improve-DBSD-methodology
OWNLINKS:  porting-problem
      translating-problem
      enhancing-problem
```

```
-%%-Emacs: dhc-text.tmp          (Fundamental)----All----------
o link selected
```

emacs: Emacs @ wurttemberg

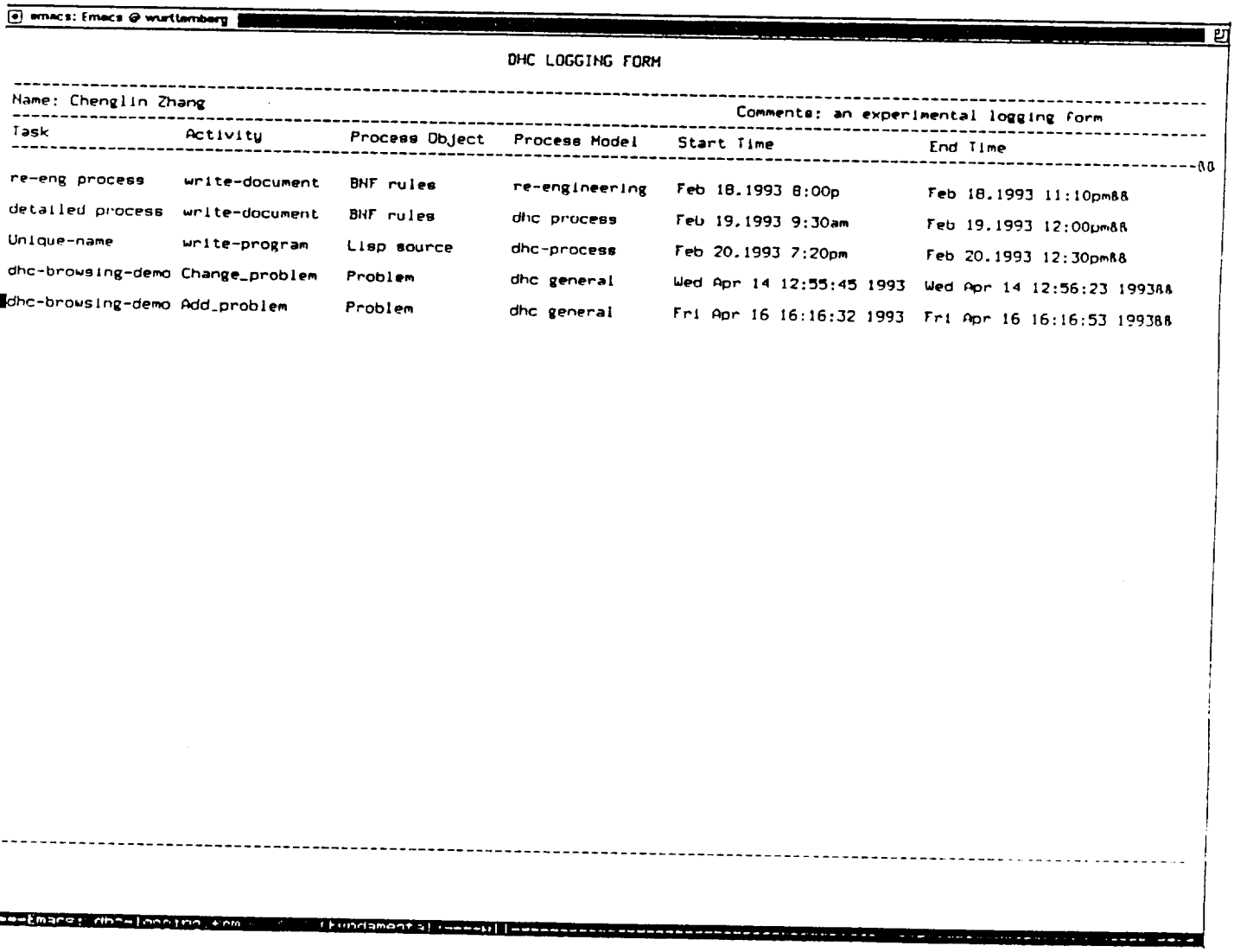
```
)BLEM: reengineering-process-model
:velop a process model for porting,translating and enhancing applications
'ERNATIVES:  1) Develop separate methodologies for:
      a. Porting to different dialects of COBOL(e.g. Honeywell to Microfocus)
      b. Enhancing applications (e.g. using SQL language instead of file oper\
ions)
      c. Translating into another language (e.g. COBOL to Ada)
   2) Develop a general methodology for everything.
CISION:  1)
STIFICATION:  The process is too little understood to fully develop a general\
ethodology for everything.
ALUATION:  We assume that the existing application is partially DHC-ed.

LINKS:  simple-porting
WNLINKS:  process-model-implementation
    process-model-notations
    get-Refinery-process-model
    get-DHC-process-model-for-Rengineering
    combine-DHC&Rfinery-process-models
```

-%%-Emacs: dhc-text.tmp (Fundamental)----Hi----

acs: Emacs @ wurttemberg

```
_EM: get-DHC-process-model-for-Rengineering
ıst the DHC process model to the porting problem
RNATIVES:
SION:
IFICATION:
UATION:
NKS:  reengineering-process-model
ILINKS:
```

```
-%%-Emacs: dhc-text.tmp          (Fundamental)----All----------------------
:uplink, d:downlink, c:current, l:latex, g:graph, f:form, p:process, i:index
```



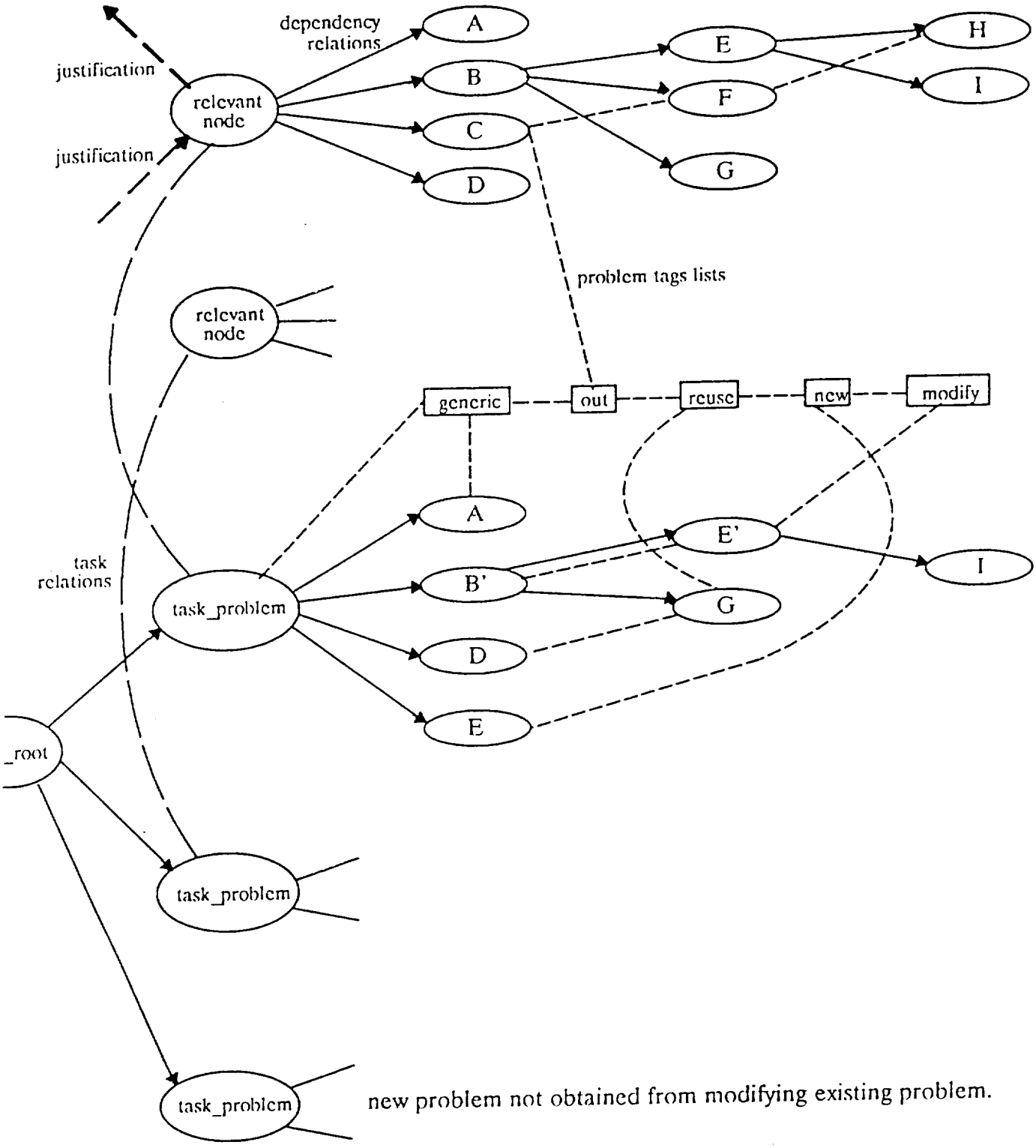
emacs: Emacs @ wurtlamberg

DHC LOGGING FORM

Name: Chenglin Zhang

Comments: an experimental logging form

Task	Activity	Process Object	Process Model	Start Time	End Time
re-eng process	write-document	BNF rules	re-engineering	Feb 18,1993 8:00p	Feb 18,1993 11:10pm&&
detailed process	write-document	BNF rules	dhc process	Feb 19,1993 9:30am	Feb 19,1993 12:00pm&&
Unique-name	write-program	Lisp source	dhc-process	Feb 20,1993 7:20pm	Feb 20,1993 12:30pm&&
dhc-browsing-demo	Change_problem	Problem	dhc general	Wed Apr 14 12:55:45 1993	Wed Apr 14 12:56:23 1993&&
dhc-browsing-demo	Add_problem	Problem	dhc general	Fri Apr 16 16:16:32 1993	Fri Apr 16 16:16:53 1993&&



new problem not obtained from modifying existing problem.

Key: task relations
problem tags lists
dependency relations
justifcation

Tentative task problem space and existing problem space.

DHC Process Model

ftware_development --> $task_{SE,MA}$ > $Understanding_{SE,MA}$ /*requirements_definition*/

task_root: task_problem > {problem} /*list_of_reusable candidates*/

Task_problem_solving > {task_problem} /*first_level_decomposition*/

{$Assessment_{MA,SE}$ > (task_root, effort, task_root.size, task_root.risk)

Change_task_decision > task_root}

Assign_resources > schedule

Transfer_task_to_problem_space > agenda

{agenda; schedule > $Solve\_problem_{SE}$ > agenda

Review_meeting > meeting_notes

Implement_meeting_decision > schedule, agenda}

DHC Process Model

oring --> Requirements_definition || Reusability_search

:ability_search --> criteria > search_space Select_node > **add_reusable_node** > task_problem.reuse_list

:_problem_solving --> task_root > ∇ {node ∈ relevant_nodes} > (Modify_existing_node > {task_problem}

{task_root > Create_additional_new_task > task_problem})

odify_existing_node --> node > **create_task_problem** > task_problem, task_relation **tag_subproblems** > node, task_problem >

{Add_new_feature | Delete_old_feature | Change_old_feature | Copy_generic

| Add_reuse /* for all nodes on reuse list}

dd_reuse --> **change_description adjust_justification adjust_reuse_task_problem add_to_reuse_list**

DHC Process Model

essment --> Calculate_Direct_Effect Calculate_Indirect_Cost *Review_Data*$_{MA}$

culate_Direct_Effect --> ∇ {task ∈ task_problem} ∇ {task_node ∈ task.generic U task.out U task.reuse}
> **calculate_node_size calculate_risk_effort** > {task_node}

∇ {task_node ∈ new} > **get_effort_risk_size_estimates** > {task_node}

∇ {task_node ∈ modify} > Assessment > {task_node}

{task_node} > **calculate_task_problem_size** > task
/* for each category add the number in the subproblem nodes
to obtain the relevant figures in the task problem node*/

{task_node} > **calculate_task_problem_effort** > task
/* this is the sum of the efforts in the subproblem list */

{task_node} > **calculate_task_problem_risk** > task
/* the sum of the risks in the subproblem nodes */

{task} > **add_up_direct_costs** > task_root

culate_Indirect_Cost --> relevant_node > **get_closure_list_justification_to_from** > ripple_list: {problem}

/* get the worst possible impact by calculating the transitive impact closure for the justification limits */

∇ {node ∈ ripple_list} > **calculate_total_node_size** > {task_problem.upper_bound}

/* add up all the metrics, # problems, #LOCS, etc, for all the nodes in the closure */

/* allow for interactive estimates */

| ∇ {node ∈ ripple_list} > (Select_node **calculate_total_node_size**) > {task_problem.lower_bound}

/* add only selected nodes to the calculation */

{task_problem} > **add_up_indirect_cost** > task_root

ACTIVITIES
(non-terminals)

Add-conditional-decision
Add-info-to-node
Add-new-feature
Add-problem
Add-unconditional-decision
Adjust-agenda
Adjust-and-add-reuse
Assign-resources-to-schedule
Calculate-Direct-Effect
Calculate-Indirect-Cost
Calculate-effect-and-costs
Change-conditional-decision
Change-old-feature
Change-problem
Change-unconditional-decision
Copy-generic
Create-additional-new-tasks
Create-and-add-new-task-problem
Customer-requirement-resolution
Delete-decision-and-replace
Delete-old-feature
Exploring-needs
Find-and-add-reusable-node
Generate-problems
Implement-meeting-decisions
Internal-development
Locate-decision
Locate-parent
Locate-problem
Make-decision-using-alternatives
Make-node-and-add-info
Modify-or-create-new-task
Modify-task-node
Pick-node-link
Read-document
Read-justification
Read-node-description
Requirements-definition
Review-agenda
Review-meeting-reports-and-progress
Review-reports-agenda-and-schedule
Select-and-read-node
Software-development
Solve-problem
Transfer-task-to-problem-space
Understand-problem-links
Understanding-needs
Visit-and-read-node

FEATURES
(terminals)

add-agenda-problem
add-justification
add-reuseable-node
add-to-generic-list
add-to-modify-list
add-to-new-list
add-to-notes
add-to-out-list
add-to-report
add-to-reuse-list
add-up-direct-costs
add-up-indirect-cost
adjust-justification
adjust-reuse-task-problem
browsing-agenda
calculate-node-size
calculate-risk-effort
calculate-task-problems-effort
calculate-task-problems-risk
calculate-task-problems-size
calculate-total-node-size
change-description
conditional-decision
copy-new-generic-task-problem
create-new-problem-node
create-new-task-problem
create-task-problem
decision
delete-agenda-problem
delete-problem
delete-task-problem
dependency-down
dependency-up
describe-alternatives
describe-problem
document-view
emacs-commands
fill-in-description
get-closure-list-justify-to-form
get-effort-risk-size-estimates
get-parent-node
give-justifications
justification-from
justification-to
key-subproblems
link-justification
locate-problem
locate-decision
locate-parent
make-decision
make-new-replacement
modify-agenda-problem
problem
produce-schedule
read-description
research-problem
review-agenda
review-report
review-schedule
scroll-view
select-alternatives
skip
switch-view
take-agenda-problem
task
terminate-at-node-closure-relevant-nodes
transfer-tentative-to-problem-space
unconditional-decision
write-and-link-documentation

non-dhc FEATURES

access-report
add-direct-costs
add-indirect-costs
add-to-report
calculate-node-size
calculate-risk-effort
calculate-task-problem-effort
calculate-task-problem-node
calculate-task-problem-risk
emacs-commands
get-effort-node-size-estimates
make-new-requirement
produce-schedule
review-schedule

ACTIVITIES
(non-terminals)

FEATURES
(terminals)

add-agenda-problem
add-justification
add-reuseable-node
add-to-generic-list
add-to-modify-list
add-to-new-list
add-to-notes
add-to-out-list
add-to-report
add-to-reuse-list
add-up-direct-costs
add-up-indirect-cost
adjust-justification
adjust-reuse-task-problem
browsing-agenda
calculate-node-size
calculate-risk-effort
calculate-task-problems-effort
calculate-task-problems-risk
calculate-task-problems-size
calculate-total-node-size
change-description
conditional-decision
copy-new-generic-task-problem
create-new-problem-node
create-new-task-problem
create-task-problem
decision
delete-agenda-problem
delete-problem
delete-task-problem
dependency-down
dependency-up
describe-alternatives
describe-problem
document-view
emacs-commands
fill-in-description
get-closure-list-justify-to-form
get-effort-risk-size-estimates
get-parent-node
give-justifications
justification-from
justification-to
key-subproblems
link-justification
locate-problem
locate-decision
locate-parent
make-decision
make-new-replacement
modify-agenda-problem
problem
produce-schedule
read-description
research-problem
review-agenda
review-report
review-schedule
scroll-view
select-alternatives
skip
switch-view
take-agenda-problem
task
terminate-at-node-closure-relevant-nodes
transfer-tentative-to-problem-space
unconditional-decision
write-and-link-documentation

Preliminary Draft: 11/18/92

ATTACHMENT #2

Primary draft of the Process

Model for porting

Process Model

Writing Transformations

```
Reengineering -> taskCU > (Porting|Enhancing|Translating) > system

Porting -> UnderstandingMA,SE > d.task_root:d.task_problem
           AssessmentMA,SE > d.task_root > Change_decisions >d.task_root
           Assign_resourcesMA,SE > d.schedule, d.agenda
           Task_problem_solvingSE > {d.task_problem}

{d.agenda,d.schedule >  Solve_problemsSE > d.agenda
                        Review_meeting > d.meeting_notes
                        Implement_meeting_decisions > d.schedule,d.agenda

Understanding -> Get_familiar

Get_familiar -> (First_pass|Additional_pass)

First_pass -> (Read_manual|Sample_target|Compile_target)

Assessment ->     /* for read_manual choice */
     source_manual:manual > read_appendixSE >
          idiosyncracies_source:idiosyncracies
     target_manual:manual > read_appendixSE >
          idiosyncracies_target:idiosyncracies
     idiosyncracies(source/target):idiosyncracies > compare_differencesSE
          > list_of_transforms:transforms
     list_or_transforms:transforms > Task_problem_solving
Task_problem_solving ->
     original_source:source > r.open > r.ast:ast
     FOR_ALL x in list_of_transforms:transforms{
             x> {write_single_transformationSE
                 test_transformationSE
                (debugSE | done)} >
                        transformation_rule_list:transformation}
     transformation_rule_list:transformation > Generate_target
```

Process Model

Using Transformation

```
:ate_target ->
 transformation_rules:transformation > run_rulesSE >
      target_source:source
 move_target_compile > (list_or_errors | clean_compile)
 FOR_ALL errors in list_of_errors{
         errors> {(write_trouble_reportSE | fix_manuallySE)}
```

Process Model Objects

Source = {lines of source code}

Manual = (reference guide, users guide, etc.)

Idiosyncracies = {language grammar rules or examples specifying machine specific implementations}

Ast = {abstract syntax tree}

Transforms = {mapping of idiosyncracies from one machine to another}

Transformation = {rule for pattern matching to convert existing pattern to new pattern}

ATTACHMENT #3

Meeting Notes

Meeting Report: 1

Meeting Date: 26 March 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Dr. Kurt Maly	Tamara Taylor
Dr. Christian Wild	
Sooshma Bokil	

A. Opening Remarks:

Dr. Maly opened the meeting at 2:00 PM. He recommended that Dr. Wild send a letter to the dean stating Ms. Taylor's status as a student. This is to avoid the question of a possible conflict of interest since she is the representative from Paramax to the Decision Based Software Design/Refinery working group.

B. Current Status Review:

No agenda was provided. The following issues/points were discussed.

1. At Dr. Wild's request, Ms. Taylor questioned Reasoning Systems as to the possibility of recording line numbers in the abstract syntax trees (ASTs) for a possible mapping between Refinery and Decision Based Software Design (DBSD). Reasoning does maintain line numbers and offered four possibilities for accessing. The line numbers are not recorded in the AST. An attribute would need to be added in order to map to the decision view.

Problem: Retaining Decision Structure through conversion to AST and back

2. Assuming the line number attribute is added, Dr. Maly questioned how the line numbers would be mapped back to the source and decision view once the transformation is done. Dr. Wild responded that DBSD will have to look at the transformation rule to see what happens and of course there will not always be a one to one mapping of line numbers to attributes as one decision can span multiple lines/nodes. He also stated that most transformations will probably be semantic therefore, decisions will remain across the board.

Problem: (child of the above) how to maintain precision in the transformation process

3. Another area of concern that was discussed

is how will the decision views be affected when something is added during the transformation. Initial thought is that this will require a manual

update to DBSD.
olem: also child of the above: how to instrument new code added by
nsformation process.

4. Areas of concern for DBSD are:

   a. How to record the decisions that went into the transformation rules themselves,

   b. Once the transformation is complete, how will the manual fixes be implemented,

   c. When new source is introduced during the transformation, what role does DBSD take.

5. Dr. Maly questioned the ability to start and stop during the transformation process and if this is possible how integrity would be maintained across the views.

6. The above discussion was in relation to adding DBSD to existing applications and to providing hooks between DBSD and Refinery. Of another issue is if DBSD is already existent in the application being transformed. If this is the case, it was reported by Dr. Wild that you would have gaps and loose precision but not to a drastic measure.

7. Discussion moved to the task at hand for Paramax. Ms. Taylor reported that there is a definite need in the industry to reuse existing code. Customers want to move to take advantage of new hardware technologies without redoing software initially. Their primary goal is to move to the new "box" then revamp using CASE technologies to optimize once there. There is no acceptance for down time on existing applications. This is the driving force behind exploring the capabilities of Refinery.

8. Dr. Maly reported that DBSD is not related to the functional view but to the decision view. This needs clarification.

9. Future meetings will be held at 1:30 PM on Thursdays with the exception of the next meeting which will be on 31 March 1992 at 10:00 AM.

Meeting Agenda

2

31 March 1992

I. Opening Remarks

   A. Old Dominion
   B. Paramax

II. Current Status Review

   A. DBSD
   B. Refinery

>
> III. Upcoming Events
>
> Next Meeting (Proposed: Thursday, 9 April 1992,
> 1:30 PM, Old Dominion University)

From wild Mon Apr 20 18:00:18 1992
Status: RO
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
[nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil])
Received: from plevin (snapper-bb.cs.odu.edu) by chrysanthemum.cs.odu.edu
(4.1/server2.4) id AA15290; Mon, 20 Apr 92 18:00:13 EDT
Received: by plevin (4.1/lanleaf2.4)
id AA00726; Mon, 20 Apr 92 18:03:36 EDT
Message-Id: <9204202203.AA00726@plevin>
References: <9204202016.AA20740@oswald.cs.odu.edu>
From: Chris Wild <wild>
To: Tamara Taylor <taylor>
Cc: wild, maly, bokil, rosca
Date: Mon, 20 Apr 92 18:03:36 EDT

Tamara Taylor writes:

> Meeting Report: 2
>
> Meeting Date: 9 April 1992
> Meeting Location: Old Dominion University
> Computer Science Department
>
> Attendees:
>
> Old Dominion — Paramax
>
> Kurt Maly — Tamara Taylor
> Christian Wild
> Sooshma Bokil
>
> A. Opening Remarks:
>
> Dr. Maly opened the meeting at 1:30 PM.
>
> B. Current Status Review:
>
> The topics on the provided agenda were discussed.
>
> 1. DBSD.
> Tammy provided the minutes and Chris restructured the problems into the DBSD format through D-HyperCase. There were a total of six problems named w1-w6. Discussion centered on these six problem/decision definitions.
>
> w1 - Reengineering.
> Kurt felt that the w1 problem encompasses too many issues and that Chris is "prettying up" because once documented you are now accountable. Kurt feels that the reengineering capability using Refinery is a given as that is what the company is paying for and should be stated as such. As for the alternatives, Kurt was not aware that Refinery is being evaluated during this process and the possibility exists for Paramax to disband with use of this tool if it is not a desirable and cost effective approach to reengineering. Tammy stated that although this is a remote possibility, the possibility does exist after seeing how well the transformation process performs and how easy the methodology developed will be to implement. The decision to use Refine is really dependent upon the answers to the following two questions:

1. Is transformation methodology appropriately mature for use by Paramax? and

2. If so, is Refine the way to go?

Tammy stated that Paramax feels relatively comfortable that Refine is the best that is available commercially for a transformation capability. She also stated that she had spoken with James Boyle of Argonne National Laboratories who has been working in the transformation arena for well over ten years. Mr. Boyle also feels that Refine is the best transformation system available commercially. He did however, send papers on the TAMPR system which is a transformation system he works on. Tammy has provided copies of these papers to Chris and both he and she are currently reviewing from a methodology standpoint. Kurt says that when we are writing problems they need to be stated as a problem and not as an assertion.

w2 - Preserve decision structure in AST.
Let's assume that Refinery is the way to go and we are proceeding along that path. W2 addresses preserving the decision structure in the AST. This requires annotating the AST. Mapping is known because Refine is a transformation system. There is therefore a tie between the lines being processed in the source code and Refine. Refine will have to be modified to mark the lines in order to determine which transformed lines came from which original lines. It was decided after much discussion that this was the best decision eventhough it will require modifying Refine. Kurt questioned whether we should institute as policy a requirement for discussion of the alternatives. This would only be feasible if it was low cost and easily accessible. One possibility suggested by Chris was to use audio to record the conversations and then you could access as necessary. It was decided that audio is not feasible without digital access and that is not readily available. It was also decided that the alternatives should just be expanded sufficiently to state their consideration.

w3 - Precision of decision transformation.
This is an active problem and has no decision as of yet.

w4 - New code added by transformation rule.
This needs further study.

w5 - Transformation rule decision views.
oblem/decisions outlined in w1-w6.
W1, w2, w5 and w6 are considered root problems and w2 has as sub problems w3 and w4.

Of issue, is how to integrate DBSD into Refinery and to realize that there is not just one decision structure. The transformation rules have decisionow how the proc affect our decisions. Tammy has the action item to outline this prior to the next meeting. Sooshma has the action item to update the w1-w6 decisions and links to the decisions prior to the next meeting. A point well made during this meeting is that there are several process/decision views tnsformation rules,

3. Decisions concerning code that didn't pass the transformation, and
4. Decisions about the entire transformation methodology itself.

Tammy also has the action item to graph these decision views prior to the next meeting. ges of DBSD on this task. They are

1. Once a transformation is performed, use DBSD to update what didn't pass the transformation rules,
2. Use DBSD for new code added during the transformation, and
3. Use Diews.

Sooshma will update the decisions and links to w1-w6.

Meeting Agenda

3

16 April 1992

I. Opening Remarks

A. Old Dominion
B. Paramax

Meeting Report :3

Meeting Date : 16 April , 1992
Meeting Location : Old Dominion University ,
Computer Science Department .

A. Opening Remarks :

Dr. Maly opened the meeting at 1.30 PM.

B. Current Status Review :

1. Dr. Wild discussed the problems W7 – W11 that went into preparing the agenda.

Coding for W8 (removing from agenda list) and W11 (adding to agenda list) has been done. Cufrently presentation problem is being looked into .

2. Dr. Maly raised the point of interproject visibility i.e. when a system is being designed using DHC as a tool , is the designer allowed to look at

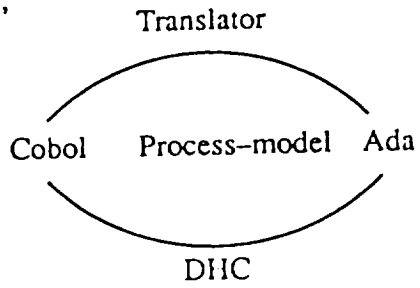
a) a whole set of decisions
b) partial set of decisions
c) no decisions

that were developed by someone else to develop the tool itself ??

3. The problem of 'how much visibility' of the above point depends on the 'context' of the projects.

Our task is to translate a specific Cobol program into Ada .

Graphically ,



a) Is the user allowed to change Process–model ?

b) Is the user allowed to look into Translator ?

c) Is the user allowed to look into DHC ?

i.e. when we are solving problems by using solutions of other problems , how much do we need to know about those solutions ?

In the next meeting we'll be discussing about

a) Translation of decision structure thru' refinery .

b) Functions of DHC .

Meeting Report: 4

Meeting Date: 23 April 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Christian Wild	Tamara Taylor
Sooshma Bokil	

A. Opening Remarks:

Dr. Wild opened the meeting at 1:30 PM.

B. Current Status Review:

The topics on the provided agenda were discussed.

1. Chris stated that we needed to discuss what was needed from Reasoning Systems to connect the DBSD structure. It was decided that the grammar is definitely needed. Tammy will provide the point of contact for this and will additionally ask Reasoning if it is possible for them to incorporate the line number attribute into the AST.

2. Tammy provided a handout on the process possibilities of Refinery and Refinery with DBSD. We went over all ten pages and made changes to page 5. The area of change is #3 which is the decision views attached to the target source including

    a. decision views attached to original source
    b. decision views attached to untransformed code
    c. decision views attached to new code generated during the transformation
    d. mapping of decision views attached to transformation rules to code.

    Please see handout for further detail. We deleted 3d as although it is still open for discussion, the possibilities of implementing this are remote due to the robustness of the requirement. We added a fifth decision view which encompasses decisions made on DHC in order to integrate with Refinery. We also discussed 3b as this is a new problem for DHC and the area of how to handle this needs to be

addressed. There are three possibilities for handling 3b. They are

1. Manually update,
2. Refinery gives some assistance, and
3. fully integrated.

Chris stated that of the three choices manually updating is unacceptable. We are not sure what assistance Refinery gives at this point. In light of this, Tammy has the action item to find out what Refinery does with untransformed code. She also has the action item to write out the process for the development methodology (#15 page 3)

Action Items:

Tammy will question Reasoning as to whether they will be able to annotate object base with line numbers.
Tammy will find out what Refinery does with untransformed code.
Tammy will provide point of contact at Kestrell/Reasoning for obtaining a university copy of Refine.

rom taylor Mon May 11 16:18:06 1992
tatus: RO
-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
["3767" "Mon" "11" "May" "92" "16:18:01" "EDT" "Tamara Taylor" "taylor" nil
eceived: from ceawlin.cs.odu.edu by chrysanthemum.cs.odu.edu
(4.1/server2.4) id AA01648; Mon, 11 May 92 16:18:01 EDT
.eceived: by ceawlin.cs.odu.edu (4.1/lanleaf2.4)
id AA28824; Mon, 11 May 92 16:18:01 EDT
lessage-Id: <9205112018.AA28824@ceawlin.cs.odu.edu>
'rom: Tamara Taylor <taylor>
To: wild
)ate: Mon, 11 May 92 16:18:01 EDT

Meeting Report: 5

Meeting Date: 7 May 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	Tammy Taylor
Chris Wild	
Sooshma Bokil	

A. Opening Remarks:

Dr. Maly opened the meeting at 1:30 PM.

B. Current Status Review:

The topics on the provided agenda were discussed.

1. We collectively discussed the Refinery/DBSD process possibilities focusing on page 5 of the handout provided by Tammy at the last meeting. The outcome of the discussion is that we are trying to solve a reengineering problem of how to incorporate Refinery and DBSd. This task initially encompasses the following:

   a. Reengineering problem - Need a system for transforming existing applications and for recording the decisions involved.
   Solution: Develop REENG a reengineering tool incorporating DBSD/Refinery.

-- Project directory /home/dhc/ReEngineering created

   b. DHC Emacs problem - While performing a., exercise DHC locating problem areas.
   Solution: Improve DHC by making appropriate modifications.

-- Project directory /home/dhc/version2/DHC-emacs

   c. Transformation problem - Automate porting Cobol code including DHC from one machine to another.
   Solution: Write Refinery transformation rules to translate cobol and DHC statements.

-- Project directory /home/dhc/version2/H2HTransformation

d. Instance of c - Verify correctness of rules written.
Solution: Instantiate c by translating

specific programs from Harris to Microfocos arena.

-- Project Directory /home/dhc/SnapPort

Action items resulting from this discussion are for Tammy to write up problems in the minutes, Chris to enter problems into DHC and to teach everyone else how to use DHC, and for Sooshma to enter the process model for solutions.

2. Tammy is to ask an additional question of Reasoning Systems concerning if there is a syntax tree construct that the unparser doesn't understand.

3. The outcome of the meeting was that we appear to have a better definition of the problem/problems we are solving and a narrower set of tasks from which to develop the transformation methodology.

Action Items:

Tammy will write up the four problems.
Chris will enter problems into DHC.
Chris will teach others how to use DHC.
Sooshma will enter process model for solutions.
Tammy will correspond with Reasoning on specific questions.

14 May 1992

I. Opening Remarks

A. Old Dominion
B. Paramax

II. Current Status Review

A. DBSD
B. Refinery

III. Upcoming Events

Next Meeting (Proposed: Thursday, 21 May 1992, 1:30
-- what happenend to this thursday?

Meeting Report: 6

Meeting Date: 14 May 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion

Kurt Maly
Chris Wild
Daniella ??

Paramax

Tammy Taylor

A. Opening Remarks:

Dr. Maly opened the meeting at 1:30 PM.

B. Current Status Review:

The topics on the provided agenda were discussed.

1. We discussed necessary revisions to a paper that will appear in a software magazine this fall. The paper describes how DHC is used in the DBSD arena. Kurt would like to see more comments in the paper about what the benefits of using DHC are. It was decided to replace the term "documentation" with "project memory" throughout the paper. It was also decided that the paper needs more hard facts. By this we mean to state that we are using DHC and how it is providing us with a multiview of our problems and decisions. Additionally, one example needs to be given and expanded throughout the paper. Kurt would also like to add tables and diagrams as they will more than likely entice interest and prompt reading of the verbiage. One high point for credit worthiness is to state claims of which we are claiming that using DHC will save you some number of man years in performing maintenance on a project. We want to stress in this paper that DHC provides a connected system from the specifications through the coding effort and the ability to retrace your decisions and their benefits and/or side effects. Daniella and Tammy will participate in updating this paper. Tammy will provide the commercial side as to the numbers of man years that are spent on maintenance etc. and Daniella will become the resident expert on the underlying program structure of DHC.

2. Discussion moved to the task Tammy is undertaking to initially use Refine to port Cobol from one machine to the other. Kurt sees no need for transformation rules for this task. He believes that the syntax tree will not change and that there is therefore no need for a rule. He also doubts that there is an unparser. Tammy of course disputes this and says that we do have an unparser and that there is a need to do a transformation rule at this stage eventhough this appears to be a simple problem. Tammy is tasked to question Reasoning about this.

The action items from last week will continue to be worked on as the problem layout is complete but instruction still needs to be given on DHC and the solution process still needs to be modeled.

Action Items:

Chris will enter problems into DHC.
Chris will teach others how to use DHC.
Sooshma will enter process model for solutions.
Tammy will correspond with Reasoning on specific questions.
Next meeting: (Proposed Thursday 21 May 1992, 1:30 PM Old Dominion University)

From taylor Fri Jun 26 11:21:29 1992
Status: RO
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
[nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil])
Received: from horsa.cs.odu.edu by chrysanthemum.cs.odu.edu
(4.1/server2.4) id AA14825; Fri, 26 Jun 92 11:21:28 EDT
Received: by horsa.cs.odu.edu (4.1/lanleaf2.4)
id AA05775; Fri, 26 Jun 92 11:20:54 EDT
Message-Id: <9206261520.AA05775@horsa.cs.odu.edu>
From: Tamara Taylor <taylor>
To: bokil, maly, rosca, taylor, wild
Date: Fri, 26 Jun 92 11:20:54 EDT

Meeting Report: 8

Meeting Date: 24 June 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly Chris Wild	Tammy Taylor

A. Opening Remarks:

Dr. Maly opened the meeting at 1:30 PM.

B. Current Status Review:

Please note that there have been no minutes provided for 6/10/92 and 6/17/92 meetings.

1. Tammy reported on her trip to Reasoning Systems for advanced training. She said she feels more comfortable with the tool but that it is a large tool with varied capabilities and there is still a lot to learn. She did perform some transformations while she was in class on code specific to her task. She additionally held the line numbers from a D_HC file and feels she will be able to transform the needed information from the D-HC files. She will be completing the transformation rules for her specific task immediately as well as looking at D-HC for where it will be useful in this project.

   Discussion proceeded to our goals for the summer which include all parties (Daniella, Soosma and Tammy) being familiar with both D-HC and Refine. Tammy is the process model and Refine person and Daniella is the D-HC person. It is not yet determined where Soosma will concentrate her efforts. Daniella will get the next iteration of D-HC up and running in this time frame as well. At the end of the summer, we will have a more concrete process model and will know how both D-HC and Refine can be utilized on a project. We will evaluate them separately and as a package.

   The agenda for the next meeting will concentrate on

reviewing the process models for the machine port and for embedding SQL statements into ported code.

The action items to be completed prior to the next meeting are:

1. Kurt, Chris and Tammy will review existing process models and update/detail accordingly.

2. Tammy will obtain email address for Chris for persons to contact that are using Refine in classes being taught at Oregon State, Naval Post Grad School and Air Force Institute of Technology (AFIT).

3. Tammy will obtain status on slip protocol connection as it needs to be completed prior to running windows across the modem.

4. Chris should provide an update on the paper which is to be published in the fall.

Meeting Agenda

11

1 July 1992

I. Opening Remarks

A. Old Dominion
B. Paramax

II. Current Status Review (Action Items)

A. DBSD
B. Refinery

III. Upcoming Events

Next Meeting (Proposed: Wednesday, 8 July 1992, 1:30 PM, Old Dominion University)

From taylor Thu Jul 2 11:44:01 1992
Status: RO
X-VM-v5-Data: ([nil nil nil nil nil nil nil nil nil]
[nil nil nil nil nil nil nil nil nil nil nil nil "^From:" nil nil nil])
Received: from penda.cs.odu.edu by chrysanthemum.cs.odu.edu
(4.1/server2.4) id AA08112; Thu, 2 Jul 92 11:43:59 EDT
Received: by penda.cs.odu.edu (4.1/lanleaf2.4)
id AA09462; Thu, 2 Jul 92 11:44:16 EDT
Message-Id: <9207021544.AA09462@penda.cs.odu.edu>
From: Tamara Taylor <taylor>
To: maly, rosca, taylor, wild, zhang_j
Date: Thu, 2 Jul 92 11:44:16 EDT

Meeting Report: 13

Meeting Date: 1 July 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	Tammy Taylor
Chris Wild	
Daniella Rosca	
Jing Yuan Zhang	

A. Opening Remarks:

Dr. Maly opened the meeting at 1:30 PM.

Please note for history purposes that these minutes represent our thirteenth meeting and that is reflected on the meeting report number. Minutes were provided and numbered correctly for meetings one through six. No minutes were provided for meetings seven through eleven. Minutes were provided for meeting twelve but were numbered incorrectly as meeting report eight. In the future minutes will be numbered according to the chronological number of our meeting just as this one is numbered thirteen.

B. Current Status Review:

Kurt provided a handout with an updated version of the reengineering problem and it's sub problems entered in D-HC and a process model of the reengineering problem using the context free grammar. Discussion revolved around the problems/model and the handout was updated accordingly. It was decided that the reengineering problem would only cover porting, enhancing and translating. It was also decided that the first pass of any of the three areas would be different than additional passes. It was additionally decided that part of the first pass is a "getting familiar with" stage. Update to format for context free grammar are that we will use activities with an agent subscript and objects will be denoted with a tool and a period (e.g. d.convention for something D-HC knows about, can modify or produce.) Daniella will update the D-HC problems and Tammy will update the process model.

The agenda for the next meeting will concentrate on reviewing the updated D-HC problems and the updated process model.

The action items to be completed prior to the next meeting are:

1. Daniella to update problem definition in C-HC.

2. Tammy to update process model.

3. Chris should provide an update on the paper which is to be published in the fall.

4. Tammy to provide updated status on email addresses for instructors of Refine.

Meeting Agenda

14

8 July 1992

I. Opening Remarks

A. Old Dominion
B. Paramax

II. Current Status Review (Action Items)

A. DBSD
B. Refinery

III. Upcoming Events

Next Meeting (Proposed: Wednesday, 15 July 1992, 1:30 PM, Old Dominion University)

Meeting Report: 14

Meeting Date: 8 July 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	Tammy Taylor
Chris Wild	
Daniella Rosca	
Jing Yuan Zhang	

A. Opening Remarks:

Dr. Maly opened the meeting at 1:30 PM.

B. Current Status Review:

It was decided that we need a process model for D-HC that is more reflective of what is existing and what is to exist in the next iteration. The current D-HC process model is an ideal model that is to be worked toward but is not reflective of the current state. Discussion centered on this topic with several action items being assigned. Additionally, we defined a process model as being "rules of interaction among the agents of change be they tools, humans, etc." We will also need D-HC to incorporate a view of the process model in the decision view so that you can assess where the process model needs to change when a change occurs in a tool/function modeled by the process model. This view also needs to be filtered according to individual needs.

The following action items resulted from this meeting:

1. Chris will provide a list of what will be added to D_HC from the existing process model and a previous functional grouping.

2. Chris will propose what functionality will be provided for next iteration and will work with Daniella and Jing Yuan on updating the process model.

3. Chris will rearrange the existing directory structure to accommodate Kurt's proposed reengineering problems/process model. He will

additionally input the problems and process model into D-HC as a demo to those of us who will be working with it.

4. Daniella and Jing Yuan will enter onto the agenda the task of printing out documentation and of filtering views. They will additionally work on the code for these two items though not prior to the next meeting.

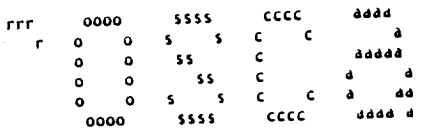
5. Tammy will update the process model according to the standard conventions and will rework again according to specific objects.

6. Everyone will keep a notebook of what they are doing in terms of activities and record the when, where, what and how long that is involved.

7. Further discussion needed on reuse of D-HC from one task to another. Will this be implemented or not?

8. **Everyone** will use D-HC and become familiar so as to make recommendations for update. Use the ~/dhc/demo directory and "reset" prior to use to get familiar with.

Note: Next meeting is Thursday, July 16, 1992, 2:30 p.m.



hu Job: stdin Date: Tue Oct 20 19:32:15 1992

ct 20 19:32 1992 standard input Page 1

From taylor Tue Jul 21 10:10:29 1992
To: maly, rosca, taylor, wild, zhang_j

Meeting Report: 15

Meeting Date: 16 July 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	Tammy Taylor
Chris Wild	
Daniella Rosca	
Jing Yuan Zhang	

A. Opening Remarks:

Dr. Maly opened the meeting at 3:00 PM.

B. Current Status Review:

Chris had listed the existing functionality of D-HC along with a proposal of what should be provided in the next iteration. The proposal was discussed and will be elaborated on at the next meeting.

It wa decided that reuse of tasks in D-HC will not be implemented in the next iteration. It was also decided that the problem and decision space in D-HC should be separate. Additionally, we need a form for recording what we are working on in order to gather statistical information while working outside of D-HC.

We will review the updated process model at the next meeting.

The following action items resulted from this meeting:

1. Chris will provide a hard copy and/or file name of the updated process model which contains existing functionality.

2. Chris will provide a hard copy and/or file name of proposed functionality for next iteration of D-HC.

3. Daniella will move all dependency and justification links related to the problem to within the problem space. These will be separate from the decision.

4. Tammy will provide a form to be reviewed at the next meeting which will aid in recording what they are working on in relation to D-HC/Refinery. This is to be used to statistical purposes.

The following action items remain from the 8 July 92

meeting:

3. Chris will rearrange the existing directory structure to accommodate Kurt's proposed reengineering problems/process model. He will additionally input the problems and process model into D-HC as a demo to those of us who will be working with it.

4. Daniella and Jing Yuan will enter onto the agenda the task of printing out documentation and of filtering views. They will additionally work on the code for these two items though not prior to the next meeting.

5. Tammy will update the process model according to

the standard conventions and will rework again according to specific objects.

8. Everyone will use D-HC and become familiar so as to make recommendations for update. Use the ~/dhc/demo directory and "reset" prior to use to get familiar with.

Note: Next meeting is Wednesday, July 22, 1992, 1:30 p.m.

Meeting Agenda

16

22 July 1992

I. Opening Remarks

A. Old Dominion
B. Paramax

II. Current Status Review (Action Items)

A. OBSD
B. Refinery

III. Upcoming Events

Next Meeting (Proposed: Wednesday, 29 July 1992, 1:30 PM, Old Dominion University)

Meeting Report: 16

Meeting Date: 22 July 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	Tammy Taylor
Chris Wild	
Daniella Rosca	
Jing Yuan Zhang	

A. Opening Remarks:

Dr. Maly opened the meeting at 1:30 PM.

B. Current Status Review:

The first topic discussed was how do we logg objects that we manipulate and the activities we accomplish, especially when we are outside DHC. Here we have to make a decision about the fact that the tool should support everything (including outside DHC).

The form we need should state the activities, the time spent on each of them, the order of activities. We have to choose between:

1. a form of the process model and checkout the steps that I am doing.
2. a list of the terminal activities from the process model. In this case I loose the sequence of activities and the objects on which I do the activities.

Another topic of the meeting was the visibility of the decisions, i.e. if we make a decision on a transformation rule do we make it visible on the target code?

Also we have discussed the subject of identifying a problem at one level and solving it at another level (the system's or the user's levels). We haven' made a decision yet on this subject.

On this subject Chris has come with an example: the variables in a cicle in FORTRAN. What to do? I can write a transformation rule for that and tell the user about it, or do it automatically (without telling anything to the users) and realize that this thing may be inefficient for a user that doesn't need this facility too much.

It should exist the possibility of filtering the views or part of a view I'm interested to see.

The following action items resulted from this meeting:

1. Daniela and Jing Yuang will continue to work on the agenda functions in order to extend the actual facilities of the system.

2. Tammy will update the form she presented as we have discussed.

eting Report: 19.

eting Date: 9/23/92
eting Location: ODU

tendees:

ODU:
Kurt Maly
Daniella Rosca
Chris Cowles
Chenglin Zhang

Paramax:

he meeting began at 2:15. Chenglin (Lin) was introduced. Names and passwords ere exchanged. NB: <cowles> <rosca> <maly> <zhang_c> <taylor>

ermissions are needed for Cowles and Zhang for the /dhc directory. Chris will see Ajay to see that it is done. Zhang will also be added to the "faculty" Email list.

Dr. Maly began a discussion on the minutes of the previous meeting. He suggested that the phrase (found under B:) "charts with problems and decisions for whatever we will develop" be changed to read "whatever we will and have developed". Over th course of this discussion, the following points were made:

-Definitions of DBSD and re-engineering (Refinery) were restated. Refinery is used in a graphical environment and is used to write rules. DBSD is ODU's proto it's another tool to be used with Refinery. We wish to use both of these in a unif environment.

-A problem: in porting source-code to another machine (say, A to B), two pe (say, the user and the porting engineer) may have different "decision views" in sou How do we support different decision structures on the same source code? For examp single LOC may have 2 decisions attatched to it; it depends on who is viewing it (t user or the engineer) as to which decision(s) are shown with this LOC.

-In the above example, we do not wish to have both operations active at the same time; i.e., only when the code is completely ported to machine B is it then tu over to the user. (Also: might the engineer sometimes need to look at the code and decision structure from the user's point of view?)

The following points & observations were also made:

-All project problems are to be entered into the problem space and agenda A COME UP.

-We need to evaluate DHC; make it more stable and useful enough. Possibly some functionality.

-Make problem descriptions more in depth from now on.

-Add to the chart what Daniella has done.

-Make and keep notes regarding conditional decisions; add to the DHC code s that we are able to backtrack decisions.

The meeting was quickly ended at 3:15 pm, as we all rushed off to the colloquium.

From rosca Wed Oct 14 10:41:16 1992
To: maly
Subject: meeting notes21
Cc: rosca, cowles, taylor, zhang_c

Meeting Report: 21

Meeting Date: 7 October 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	
Chris Wild	
Daniella Rosca	
Chris Cowles	
Chenglin Zhang	

A. Opening Remarks:

Dr. Maly opened the meeting at 1:30 PM.

B. Current Status Review:

First we have discussed the deliverables for this phase of the project. They will be: the paper from '91, the paper from '92(IFIP), the viewgraphs for boths and the meeting notes.

In this meeting we have discussed the objectives of our project for the following months to come. Basically, we have addressed the topic of the integration of DHC and Refinery. We have two possibilities: make them loosely coupled, seeing each other like a black box that executes its job sequential in time with respect to the other tool or make them tightly coupled.

To answer this question we have to answer first the question: are the transformation rules built using DHC or Refinery? If we use DHC for writing the transformation rules we will use Refinery afterwards as a compiler for the transformation rules.

For a tightly coupled version we will need to embed DHC in Refinery, to consider each of the Refinery functions as black boxes and wrap them in DHC functions, if they are sufficiently small. Also we would need that this functions be noninteractive so that we can have the control of the user actions at the DHC level. For this we would need a deep understanding of the source of Refinery, to figure out how to make the link with DHC. We would need from Paramax a detailed list of capabilities and functions of Refinery.

Dr. Wild said that from the discussions with Tammy resulted that from the past and current experience it seems to be no need for a tightly coupled version. Anyway we need to ask them again and to thoroughly analize which are the gains from writing the transformation rules in DHC and which are the gains from writing them in Refinery.

Another question that we have to ask Paramax is if they want to support also the reverse porting, in the case when is needed an enhancement in the ported program. Do they want to maintain the 2 versions of the program consistent, so we should have the capability of going back and forth between the 2 versions of a program, or once we have done the porting, the older version will not be considered anymore.

From the discussion resulted the following guide lines for the future development of DHC:

- add the filters for different views:
  - for porting engineers
  - for enhancing engineers
  - for project managers
- develop new evaluation methods, new measures to have automatic statistics.
- to enhance the existing process model.

Meeting Agenda

21

7 October 1992

I. Opening Remarks

A. Old Dominion
B. Paramax

II. Current Status Review (Action Items)

A. DBSD
B. Refinery

III. Upcoming Events

Next Meeting (Proposed: Wednesday, 14 October 1992, 1:00 PM, Old Dominion University)

cowles Wed Oct 14 14:49:36 1992
osca
ct: last meeting

nclosing the minutes i've written so far. Do you think there is more
i should add? Some things i admit, i just didn't get. I don't expect
o write the minutes for me; but let me know if you have any comments.
lso, what waas it the maly was calling the Navy division that Paramax
s with? was it NAVMEX or something?

Meeting Report: 22

Meeting Date: 14 October 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	
Daniella Rosca	
Chris Cowles	
Chenglin Zhang	

. Maly opened the meeting at 1:10 PM.

ris and Lin are currently 1) reading the .el files, 2) continuing with
SP, and 3) learning to run the DHC demo. Dr Maly would like us, within
o weeks, to 1) be more familiar with DHC, 2) have a reasonable idea of the
de structure, and 3) be close to being able to make modifications in DHC.

aniella is to put together the report to Paramax. She needs to 1) gather
he material, 2) start to write a cover letter, and 3) have a particular
action-list". We will request of Tammy Taylor a list of terminals and non-
erminals in Refinery. November first is the target date to submit this
eport.

ur next meeting will be on Tuesday (Oct. 20th) at 11 am. due to a conflict.
or our other regular Wednesday meetings, the 1 pm time is firm, at least for
he time being.

Paramax's task (ie., Tammy's) is to use Refinery to port code. Our tasks is
to show that we can take their output in whatever form, and complete the
process of transforming it into a complete second form of the code (eg., in
COBOL). We want to show that DHC is a useful tool for porting - to complete
the transportation of that (COBOL) program.

As regards our marketing efforts, we would like to hear from users (Paramax,
Navy) about such things as knowing how many languages and ships are involved.

In writing down all Problems in the overall Problem Space: do we face the
problem of knowing whether or not this Problem has already been defined in
the Problem Space? Is a Problem part of a larger Problem Structure? Where
does it fit into the Problem Space? It seems that one has to know the entire
Problem Space in order to know where this Problem fits in. We note that
every problem being solved is tied to a requirement.

Since we all seem at this point to have well-defined tasks to do, the
meeting was ended at 2:10 pm.

From rosca Tue Oct 27 22:20:15 1992
Received: from ramses.cs.odu.edu by chrysanthemum.cs.odu.edu
(4.1/server2.4) id AA26895; Tue, 27 Oct 92 22:20:13 EST
Received: by ramses.cs.odu.edu (4.1/lanleaf2.4)
id AA00416; Tue, 27 Oct 92 22:28:10 EST
Message-Id: <9210280328.AA00416@ramses.cs.odu.edu>
Date: Tue, 27 Oct 92 22:28:10 EST
From: Daniela Rosca <rosca>
To: maly
Subject: meet23.notes
Cc: rosca, cowles, zhang_c
Status: R

Meeting Report: 23

Meeting Date: Oct. 21 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	
Daniella Rosca	
Chris Cowles	
Chenglin Zhang	

A. Opening Remarks:

Dr. Maly opened the meeting at 1:00 PM.

B. Current Status Review:

We looked over the report and cover letter prepared by Daniela for Paramax. We need to add a table of contents and a complete chart of the Reengineering problem with the corespondence between problems and descriptions.

In the report we have to put explicitly references to the attachements of the document and to have separate chapter for each main topic.

We have also discussed the specific questions to ask Paramax about. They concern mainly market information needed in taking our decisions.

The following action items resulted from this meeting:

1. put togheter the chart of the Reengineering problem.
2. upgrade the report according to the chart.
3. ask Tammy all the information necessary for completing items 1 and 2.

Meeting Agenda

23

Oct.21 1992

I. Opening Remarks

A. Old Dominion
B. Paramax

II. Current Status Review (Action Items)

A. DBSD
B. Refinery

III. Upcoming Events

Next Meeting (Proposed: Wednesday, Oct. 28 1992, 1:00 PM, Old Dominion University)

om zhang_c Wed Nov 11 12:44:31 1992
: taylor, cowles, rosca, maly
oject: meeting note 24, final

Meeting Report: 24

Meeting Date: 28 October 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	
Daniella Rosca	
Chris Cowles	
Chenglin Zhang	

r. Maly opened the meeting at 2:15 PM.

aniella, Chris and Lin prepared a chart of the reengineering problem space hich we had defined so far for the meeting. The following discussion was ased on the chart.

r. Maly pointed out that some problems, such as market future and the process f reenigneering to Paramax, were missing and reemphasized the importance that very problem, whether it is currently linked or not, should go into the chart.

he evaluation of dhc, Refinary, and process model is not the problem of re-engineering. As far as reengineering is concerned, we have to study its methodology, process model, and supporting tools. We have to make a decison on such alternatives as 1) use dhc alone; 2) use Refinary alone; 3) use them both. If we choose the last alternative, we have to decide if they are looselycoupl

In a loosely-coupled approach, we have to 1)upgrade dhc; 2)get more knowledge on Refinary; and 3)determine the interface between dhc and Refinary. Here we --Moron Refinary; and 3)determine the interface between dhc and Refinary. Here we need more information from Tammy:
1) reengineering process in Paramax;
2) pros and cons of Refinary.

We spent some time on how to reuse solutions if the coming problem is the same as or similar to a problem in the problem space. Cut-and-paste seems to be a good strategy. Anyway, we have to make out how to build the reuse machanism into the dhc or Refinary process model.

We reached the following convensions for our future charts:
1) The text in every node of the chart should contain the name of the corres- po reference between the chart and .pd/.dd files.
2) The up and down links in decisions (UL and DL) will be dropped out because all the information will be organized around the problem space and there are in fact no links between decision nodes.
3) We do not have to require that every problem node has a corresponding decision node. In fact, we make a decision only when there are several alternatives.
4) We may add some mark symbols to problem nodes to indicate their solution status.

Some problems remain:
1) How to generate unique identities for problems and decisions. The current practice in dhc is to use the first character of login name with a number. This should be changed.
2) How to show output of problem solving in the chart.

3) How to sort the problem and decision spaces according to some specific criteria.

Because there are some things unclear to us about market and Paramax, we will concer chart. We will have an upgrated chart for the next meeting.

The meeting was ended at 3:50 pm.

Meeting Report: 25

Meeting Date: November 13, 1992
Meeting Location: Old Dominion University
Computer Science Department

Attendees:

Old Dominion	Paramax
Kurt Maly	Tammy Taylor
Daniella Rosca	
Chris Cowles	
Chenglin Zhang	

Dr. Maly opened the meeting at 9:30 am.

The focus of this meeting was to review and update the Chart and Report to be sent to Paramax.

For the Report, a Table of Contents is needed; previous meeting notes and the Chart are to be included. In the Chart, our problem is the DBSD paradigm and what we are doing for the Paramax Reengineering problem. Paramax problems include: what's a reasonable tool for porting/enhancing/transformation, and the need of a Market Study to decide how much to emphasize each. We have chosen, as of now - due to the unavailability of a Market Study, DHC and Refinery, loosely coupled.

As for the Chart's alternatives, we will add only the major ones. We also need to shade in (in xfig) all the outputs. We will "freeze in" today's changes in the Chart as part of the overall Report.

Some items not yet addressed in the Chart include: 1) using it as a quick reference to problem decisions, and 2) adding problem names in the existing boxes to be used as an index. All of the attachments for the Report are ready.

Our questions for Tammy / Paramax include: Are there any Reengineering contracts as of now? How big are they? How many LOC are involved? What are the problems in the other solutions and what are their characteristics?

As far as the actual submission to Paramax is concerned, an invoice is to be sent under separate cover and should reference the deliverable.

Tammy will be here on Tuesday to help Daniella with the Chart; we will meet again on Wednesday at 1:30 to make final adjustments; the package will be sent on Thursday, November 19th, 1992.

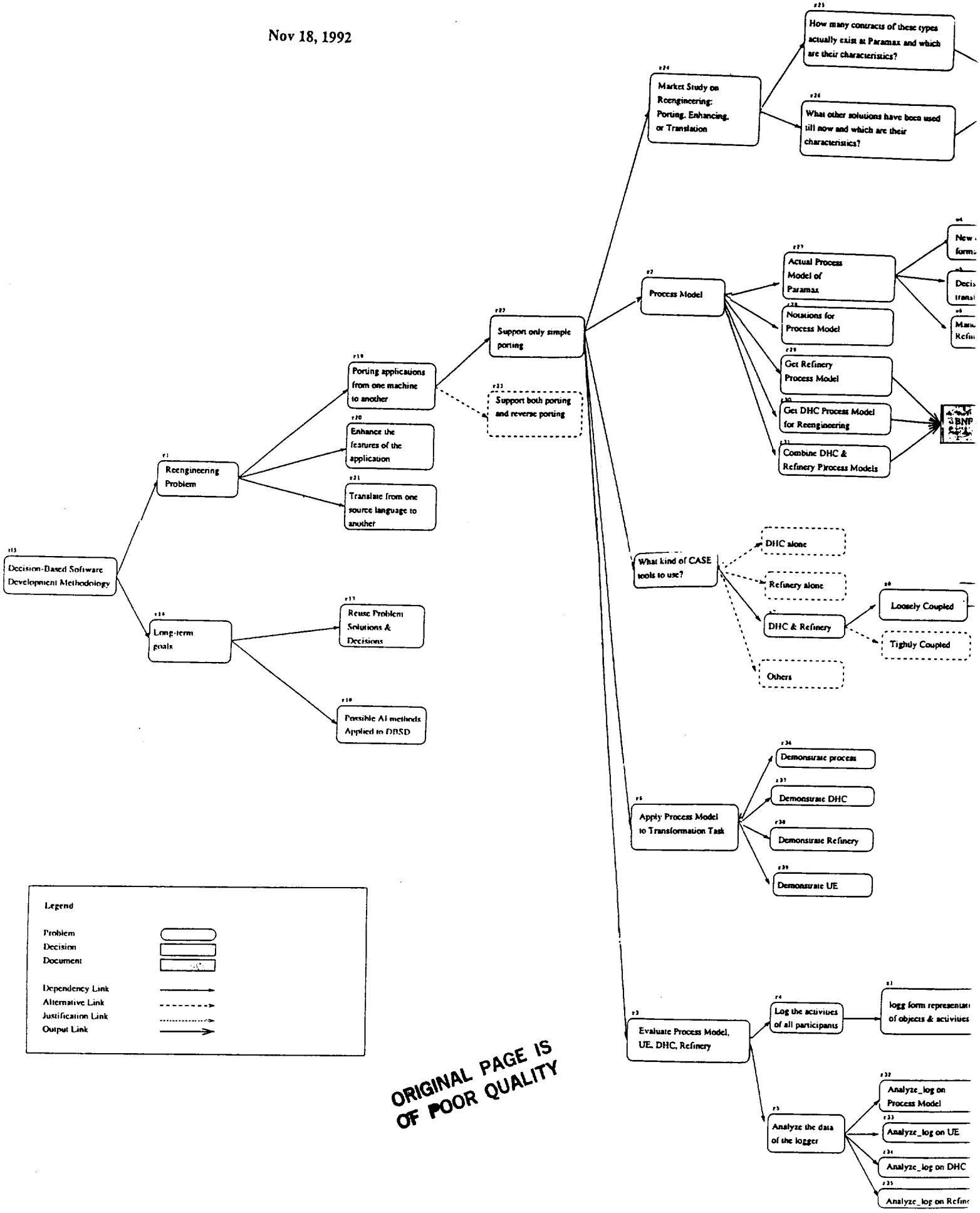
The meeting was ended at 10:30 am.

ATTACHMENT #4

Primary draft of the chart representing the problems involved by the reengineering process

Chart of the Problem Space for Reengineering

Nov 18, 1992



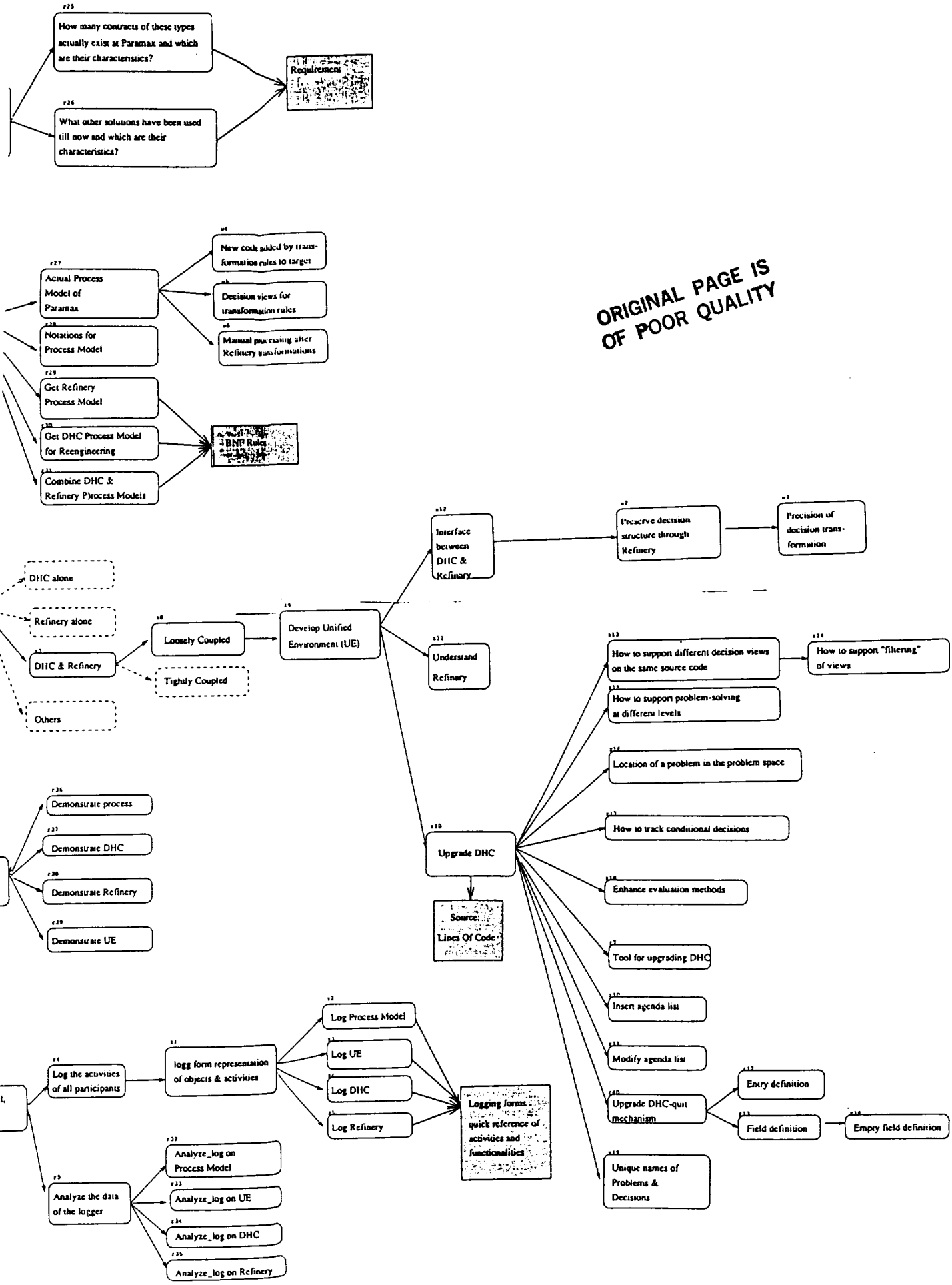
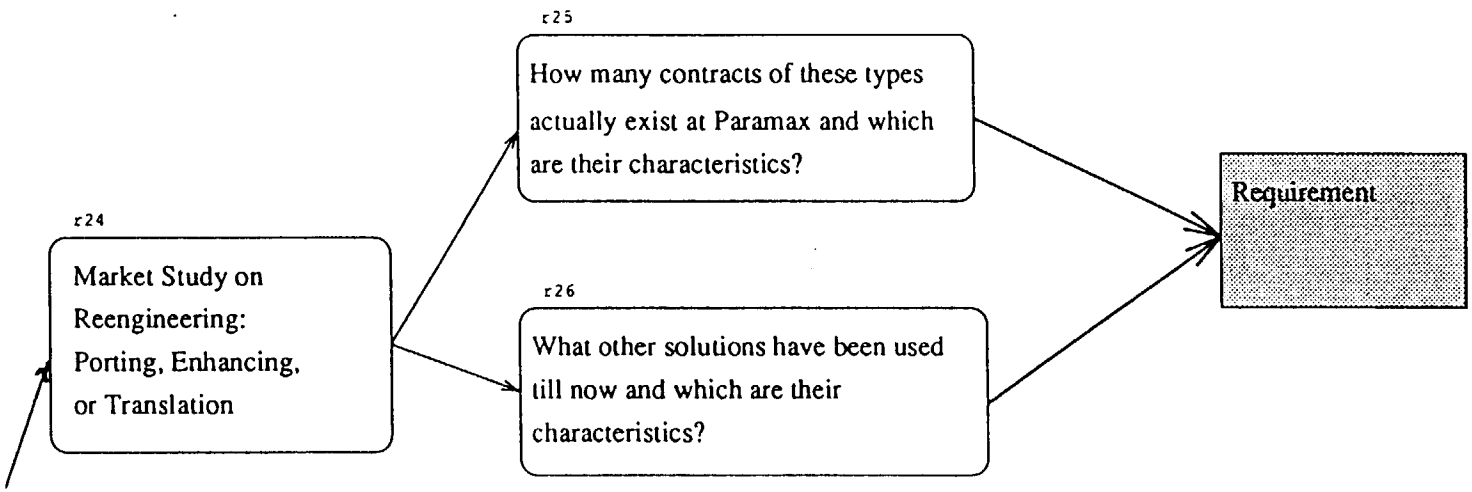
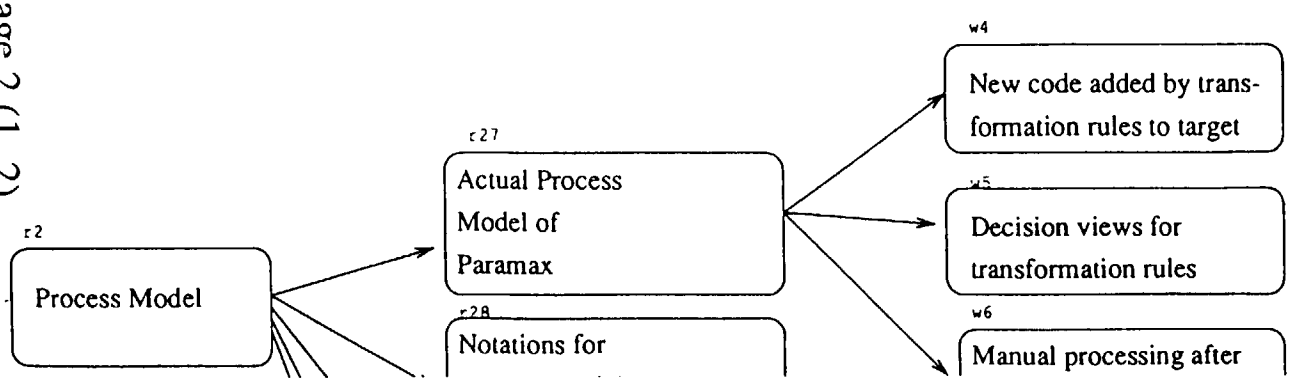
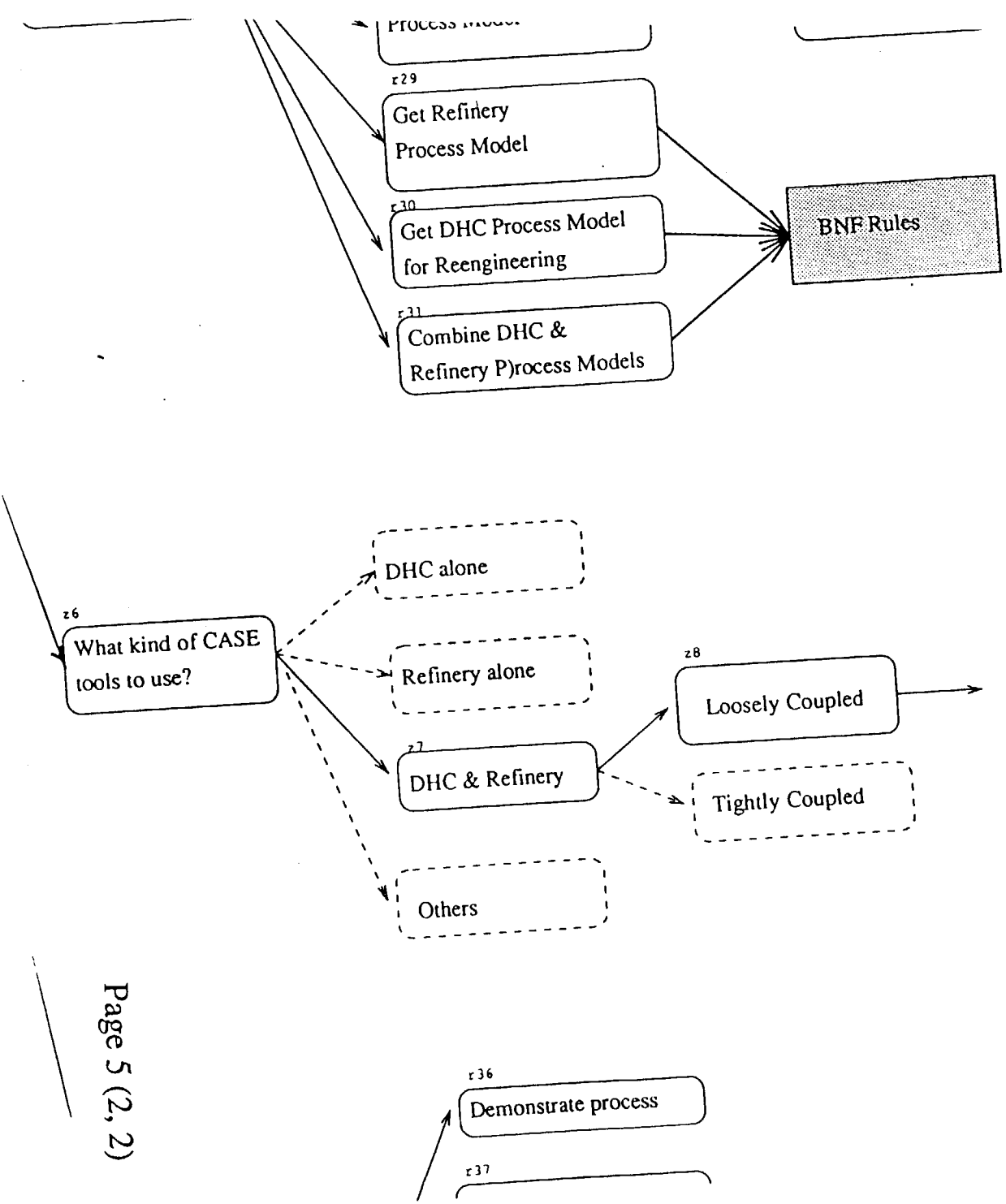


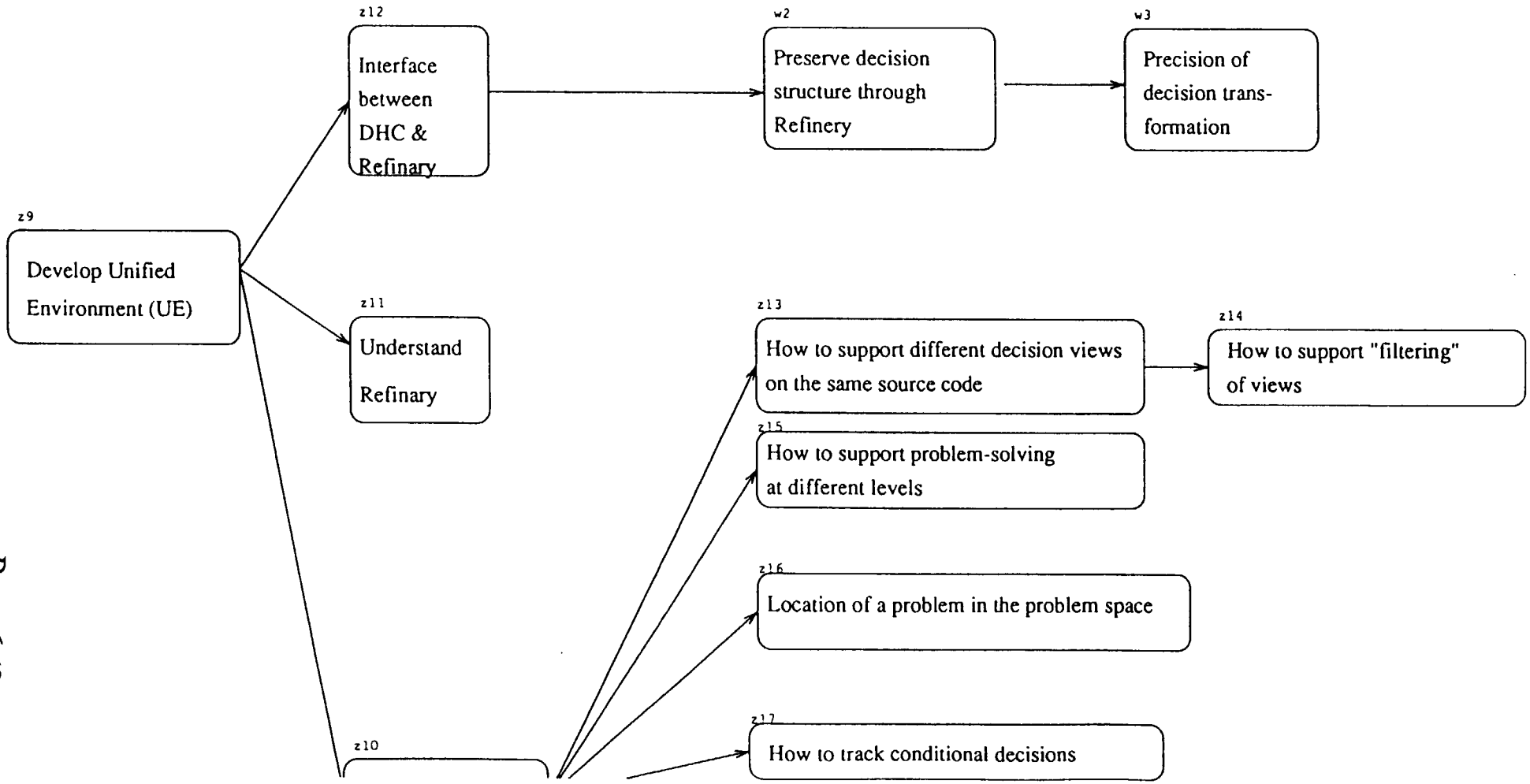
Chart of the Problem Space for Reengineering

Nov 18, 1992

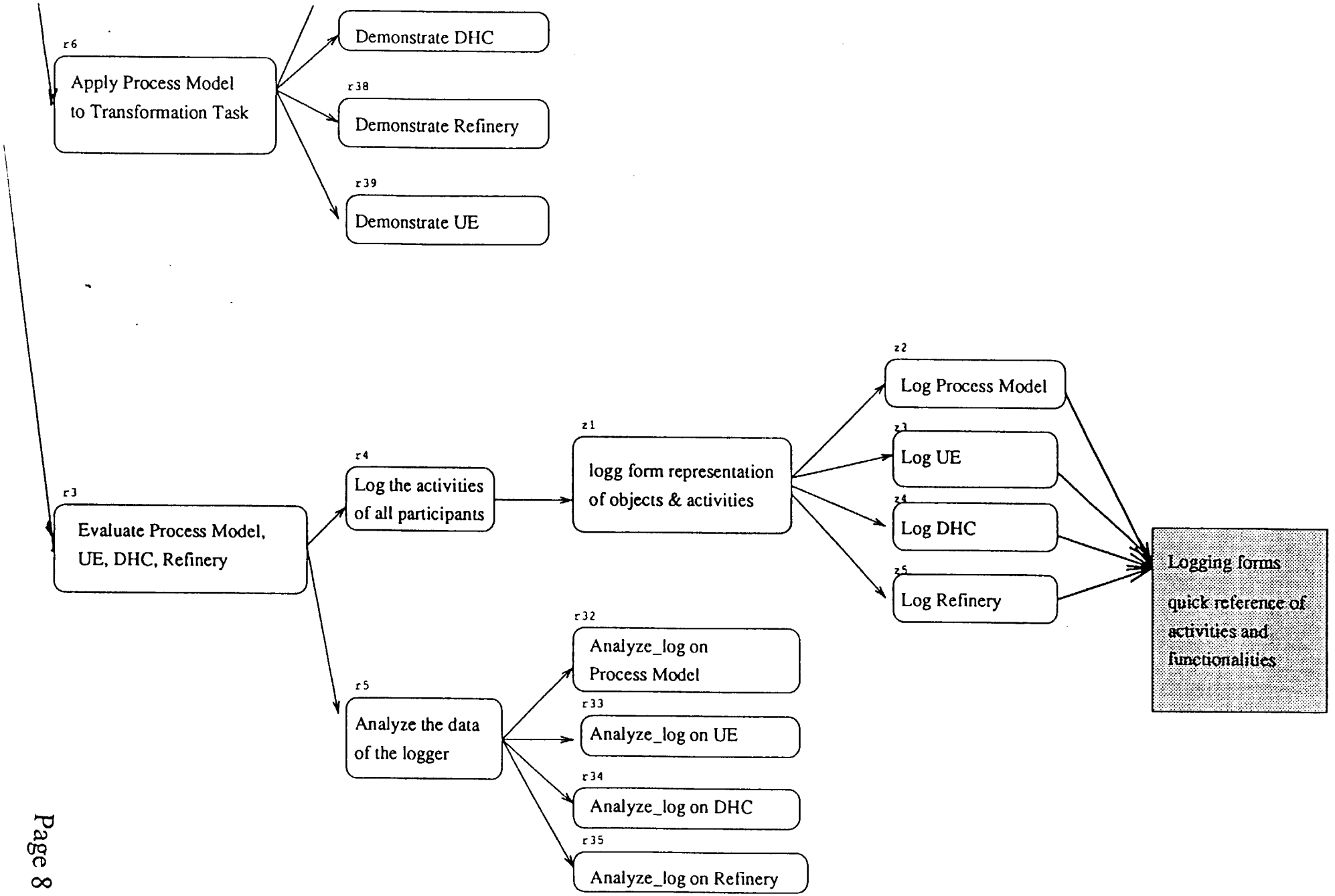

r25
How many contracts of these types actually exist at Paramax and which are their characteristics?
r24
Market Study on Reengineering: Porting, Enhancing, or Translation
r26
What other solutions have been used till now and which are their characteristics?
Requirement

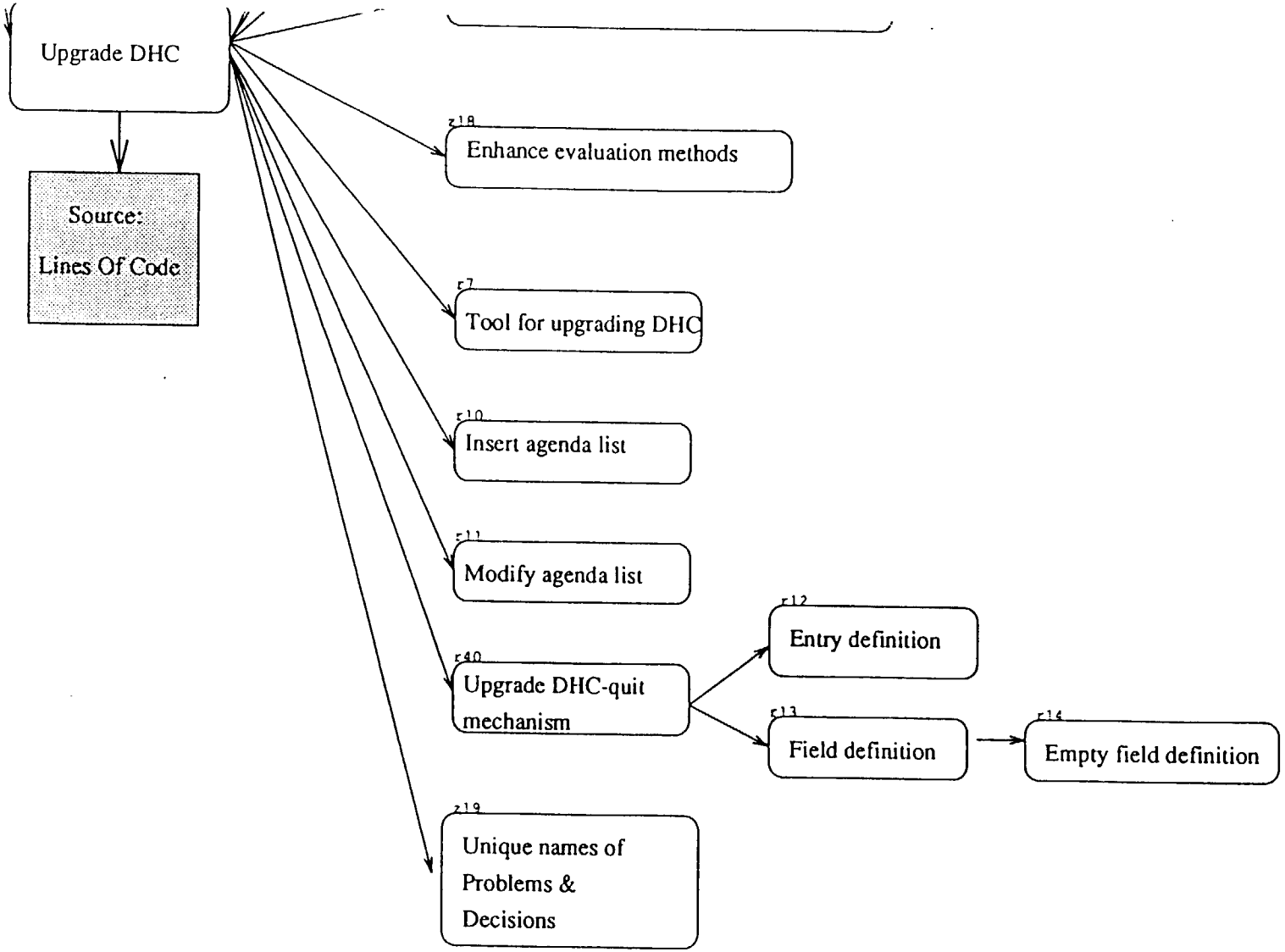

w4
New code added by trans-formation rules to target
r27
Actual Process Model of Paramax
w5
Decision views for transformation rules
r2
Process Model
r28
Notations for
w6
Manual processing after


r29
Get Refinery
Process Model
r30
Get DHC Process Model
for Reengineering
r31
Combine DHC &
Refinery P)rocess Models
BNF Rules
z6
What kind of CASE
tools to use?
DHC alone
Refinery alone
z7
DHC & Refinery
z8
Loosely Coupled
Tightly Coupled
Others
r36
Demonstrate process
r37


z9
Develop Unified Environment (UE)
z12
Interface between DHC & Refinary
w2
Preserve decision structure through Refinery
w3
Precision of decision trans-formation
z11
Understand Refinary
z10
z13
How to support different decision views on the same source code
z14
How to support "filtering" of views
z15
How to support problem-solving at different levels
z16
Location of a problem in the problem space
z17
How to track conditional decisions

Legend
Problem
Decision
Document
Dependency Link
Alternative Link
Justification Link
Output Link


r6
Apply Process Model to Transformation Task
Demonstrate DHC
r38
Demonstrate Refinery
r39
Demonstrate UE
r3
Evaluate Process Model, UE, DHC, Refinery
r4
Log the activities of all participants
z1
logg form representation of objects & activities
z2
Log Process Model
z3
Log UE
z4
Log DHC
z5
Log Refinery
Logging forms
quick reference of activities and functionalities
r5
Analyze the data of the logger
r32
Analyze_log on Process Model
r33
Analyze_log on UE
r34
Analyze_log on DHC
r35
Analyze_log on Refinery


Upgrade DHC
Source:
Lines Of Code
r18
Enhance evaluation methods
r7
Tool for upgrading DHC
r10
Insert agenda list
r11
Modify agenda list
r40
Upgrade DHC-quit mechanism
r12
Entry definition
r13
Field definition
r14
Empty field definition
r19
Unique names of Problems & Decisions

eliminary Draft: 11/18/92

ATTACHMENT #5

DHC problem and

decision spaces

PROBLEM SPACE re. pd

&&r1 reengineering_problem
P: Develop a methodology and supporting tools to port applications from a source
UL: apply_and_improve_DBSD_methodology
DL: porting_problem
DL: translating_problem
DL: enhancing_problem

&&r2 reengineering_process_model
P: Develop a process model for porting,translating and enhancing applications
UL: simple_porting
DL: process_model_implementation
DL: process_model_notations
DL: get_Refinery_process_model
DL: get DHC_process_model_for_Rengineering
DL: combine_DHC&Rfinery_process_models

&&r3 evaluation_problem
P: validate the decision in reengineering_problem
UL: simple_porting
DL: logger
DL: statistical_analizer

&&r4 logger
P: Log the activities (as defined in process model) of all participants.
UL: evaluation_problem
DL: logging_form_representation

&&r5 statistical_analizer
P: Analize the data of the logger
UL: evaluation_problem
DL: analyze_log_on_Process_Model
DL: analyze_log_on_Unified_Environment
DL: analyze_log_on_DHC
DL: analyze_log_on_Refinery

&&r6 apply_process_model_to_transformation_task
P: a. port a SNAP COBOL program running on a Honeywell to a UNIX box running Micr
b. replace the file operations with an equivalent database language.
UL: simple_porting
DL: demonstrate_process
DL: demonstrate_DHC
DL: demonstrate_Refinery
DL: demonstrate_Unified_Environment

&&w2 Preserve_decision_structure_in_AST
P: If refinery is used to transform one source code document into another, then any decision structure associated with the first document needs to be transferred to the second.
UL: interface_between_DHC&Refinery
DL: Precision_of_decision_transformation

&&w3 Precision_of_decision_transformation
P: How is the precision of decision structure maintained through the Refinery transformation. Since the AST is not line oriented, the decision view don't map one for one on the AST
UL: Preserve_decision_structure_in_AST
DL:

&&w4 new_code_added_by_transformation rule
P: How is new code added by transformation rule to be instrumented for its decision structure? It is possible that this new code solves a problem of differences between platforms or compilers (an accidental difference by Fred Brooks classification).
UL: process_model_implementation

DL:

&&w5 transformation_rule_decision_views
P: How to record the decisions involved with defining the transformation rules themselves.
UL: process_model_implementation
DL:

&&w6 manual_after_transformation
P: How to support manually processing that occurs after the transformation. Also how does the programmer understand what the transformation system has done?
UL: process_model_implementation
DL:

&&r7 tool_for_upgrading_DHC
P: What tool to use for DHC?
UL:upgrade_DHC
DL:

&&r8 robustness_of_DHC
P: What criteria should we use for choosing a tool for DHC?
UL:
DL:

&&r9 order_of_objects_editing
P: How to edit 2 objects in order: describe problem and agenda object?
UL:
DL:

&&r10 insert_agenda_list
P: Insert a problem into the agenda list.
UL: upgrade_DHC
DL:

&&r11 modify-agenda-list
P: modify entries of a problem in the agenda list to reflect the status of proble
UL: upgrade_DHC
DL:

&&r12 entry_definition
P: Entry definition in DHC files.
UL: upgrade_DHC_quit
DL:

&&r13 field_definition
P: Identification of fields in an entry in DHC files.
UL: upgrade_DHC_quit
DL: empty_field_definition

&&r14 empty_field_definition
P: Which is the definition of an empty field?
UL: field_definition
DL:

&&15 apply_and_improve_DBSD_methodology
P: apply DBSD methodology to various applications and eventually improve it, as a

&&r16 long_term_goals
P: Reuse problem, include some AI techniques, etc.
UL: apply_and_improve_DBSD_methodology

DL: reuse_problem.

&&r17 reuse_problem
P: How to reuse actual solutions from our problem space to solve new problems?
UL: long_term_goals

&&r18 AI_applications
P: Apply some AI methods, techniques to DBSD
UL: long_term_goals

&&r19 porting_problem
P: porting applications from one machine to another
UL: reengineering_problem
DL: simple_porting

&&r20 enhancing_problem
P: enhancind the features of an application
UL: reengineering_problem

&&r21 translate_problem
P: translate from a language A to another language B, on the same machine
UL: reengineering_problem

&&r22 simple_porting
P: support onlu simple porting from one source machine to a target machine
UL: porting_problem
DL: market_study
DL: reengineering_proces_model
DL: CASE_tool_for_reenginerring
DL: apply_process_model_to_transformation_task
DL: evaluation_problem

&&r23 complex_porting
P: support both porting and reverse porting
UL: porting_problem

&&r24 market_study
P: we need more information from a market study done by the Paramax
personnel to gide our efforts into the direction desired by paramax.
UL: simple_porting
DL: reengineering_contracts&characteristics
DL: reengineering_solutions&characteristics

&&r25 reengineering_contracts&characteristic
P: how many reengineering contracts exist at Paramax and which are their characteristics?
UL: market_study

&&r26 reengineering_solutions&characteristics
P: what other solutions have been used till now for the reengineering projects and which are their characteristics
UL: market_study

&&r27 process_model_implementation
P: issues in the actual implementation of the process model of Paramax
UL: reengineering_process_model
DL: new_code_added_by_transformation rule
DL:transformation_rule_decision_views
DL: manual_after_transformation

&&r28 notations_for_process_model
P: notations to use in a formal specification of the process model
UL: reengineering_process_model

&&r29 get_Refinery_process_model
P: get the process model for Refinery use in this problem

reengineering_process_model

get DHC_process_model_for_Rengineering
adjust the DHC process model to the porting problem
reengineering_process_model

combine_DHC&Refinery_process_models
get the global process model for the porting problem where we have used
the Refinery and DHC
: reengineering_process_model

2 analyze_log_on_Process_Model
analyze the data obtain with the logging forms on Process Model
: statistical_analizer

33 analyze_log_on_Unified_Environment
: analyze the data obtain with the logging forms on Unified_Environment
L: statistical_analizer

34 analyze_log_on_DHC
: analyze the data obtain with the logging forms on DHC
L: statistical_analizer

:35 analyze_log_on_Refinery
?: analyze the data obtain with the logging forms on Refinery
JL: statistical_analizer

r36 demonstrate_process
P: develop a prototype process model of the DHC methodology and the
ıvironment
UL: apply_process_model_to_transformation_task

ır37 demonstrate_DHC
P: develop an upgraded DHC prototype to determine the value of DHC
UL: apply_process_model_to_transformation_task

&r38 demonstrate_Refinery
P: demonstrate the efficiency of using Refinery for this type of problems
UL: apply_process_model_to_transformation_task

&&r39 demonstrate_Unified_Environment
P: demonstrate the utility of building a unified environment fron the
interaction of Refinery and DHC
UL: apply_process_model_to_transformation_task

&&r40 upgrade_DHC_quit
P: upgrade DHC_quit with the updating of the agenda list
UL: upgrade_DHC
DL: entry_definition
DL: field_definition

&&z1 logging-form-representation
P:How to represent the login forms?
UL:logger
DL:log-PM, log-UE, log-DHC, log-Refinary

&&z2 log-PM
P:How to represent login form for the porting Process Model?
UL:logging-form-representation
DL:

&&z3 log-UE
P:How to represent login form for the United Environment?
UL:logging-form-representation
DL:

&&z4 log-DHC
P:How to represent login form for DHC?
UL:logging-form-representation
DL:

&&z5 log-Refinary
P:How to represent login form for Refinary?
UL:logging-form-representation
DL:

&&z6 CASE-tools
P:What kind of CASE tools will be employed to support the porting?
UL:simple-porting
DL:dhc-refinary-integration

&&z7 dhc-refinary-integration
P: How to integrate DHC and Refinary?
UL:CASE-tools
DL:loosely-coupled

&&z8 loosely-coupled
P:How do we build a loosely coupled system for the porting?
UL:dhc-refinary-integration
DL:Unified-Environment

&&z9 Unified-Environment
P:What should we do to build a Unified Environment?
UL:loosely-coupled
DL:upgrade-DHC, understand-Refinary, interface-between-DHC-Refinary

&&z10 upgrade-DHC
P: How to make DHC more robust and stable enough?
UL:Unified-Environment
DL:decision-views, problem-solving-levels, problem-locating,
conditional-decisions,dhc-evaluation-enhancing,unique-names,

&&z11 understand-Refinary
P: We should have a good understanding about Refinary before we could integrate DHC and Refinary.
UL:Unified-Environment
DL:

&&z12 interface-between-DHC-Refinary
P: How do we build the interface between DHC and Refinary in a loosely-coupled Unified Environment?
UL:Unified-Environment
DL:

&&z13 decision-views
P: In porting source-code to another machine (say, A to B), two person (say the user and the porting engineer) may have different "decision views". How do we support different decision structures on the same source code?
UL:upgrade-DHC
UD:view-filtering

&&z14 view-filtering
P:It should exist the possibility of filtering the views or part of a view I'm interested to see. We should support view-filtering for porting engineer, project manager, and other users.
UL:decision-views
DL:

&&z15 problem-solving-levels
P:How do we support problem-solving at different levels? We should be able to identify a problem at one level and solving it at another

level (the system's or the user's levels).
UL:upgrade-DHC
DL:

&&z16 problem-locating
P: In writing down all Problems in the overall Problem Space: do we face the problem of knowing whether or not this problem has already been defined in the Problem Space? Is a Problem part of a larger Problem Structure? Where does it fit into the Problem Space? It seems that one has to know the entire Problem Space in order to know where this Problem fits in.
UL:upgrade-DHC
DL:

&&z17 conditional-decisions
P: Make and keep notes regarding conditional decisions; add to the DHC code so that we are able to backtrack decisions.
UL: more-dhc-fuctionality
DL:

&&z18 dhc-evaluation-enhancing
P: How to enhance DHC evaluation methods?
UL:
DL:evaluation-methods

&&z19 unique-names
P: How to generate unique identities for problems and decisions. The current practice in dhc is to use the first character of login name with a number. This is too weak and should be changed.
UL:upgrade-DHC
DL:

&&

DECISION SPACE re.dd

&&r1 reengineering_problem
A1: Develop a fully automated, non-interactive system for specific cases.
(for example, an Expert System for transforming a COBOL program)
A2: Develop an expert system using only DHC.
A3: Develop an interactive system using both DHC and Refinery.
A4: Non-automated system for a specific case.(like "awk")

D: A3) Develop a Reengineering_process_model, a running, effective DHC_prototype, make Refinery part of the environment, and link DHC and Refinery into a Unified_environment.

J: Dictated by :
a. the availability of DHC and Refinery; it will be a matter of evaluation
b. validation of the above decision
UL: reengineering_problem
DL:
C:

&&r2 reengineering_process_model
A1: Develop separate methodologies for:
a. Porting to different dialects of COBOL(e.g. Honeywell to Microfocus)
b. Enhancing applications (e.g. using SQL language instead of file operatio
c. Translating into another language (e.g. COBOL to Ada)
A2: Develop a general methodology for everything.

D: A1)
J: The process is too little understood to fully develop a general methodology fo
UL:reengineering_process_model
DL:
C: We assume that the existing application is partially DHC-ed.

&&r3 evaluation_problem
A1: Have separate Logger and Statistical_analyzer for reengineering_process_model
A2: Have one for both applications.

D: A2) Develop one Logger and one Statistical_analyzer and apply them to solving
J:
UL: evaluation_problem
DL:
C:

&&r4 logger
A1: Keep a notebook of activities, tool, feature, time start, end, products (sour
A2: Collect information by instrumenting DHC,Refinery, project accounts, UNIX.
A3: Combination of 1 and 2.

D: A3)
J:
UL:
DL:
C:

&&r5 statistical_analizer
A1: List of features(commands) and their use (frequency, duration).
A2: Size of various parts of documentation (decisions, rules, source code)
A3: Amount of new parts vs. changes in existing ones.

D:
J:
UL:
DL:
C:

&&r6 apply_process_model_to_transformation_task
A:
D:

re.dd

J:
UL:
DL:
C:

&&w2 Preserve decision structure in AST
A:
1) Manually reconstruct the decision structure
2) transfer the decision structure into the AST (Abstract Syntax Tree)
3) semi-automatic match of old and new to transfer
D: A2
J: A1 is too labor intensive, A2 should be possible since information about the line numbers if available during the parse. In fact they seem to use this information in linking the AST to the source code. If the line numbers were kept in the AST, then the decision views would also be known (LINK to decision to have as the least granularity of a decision the line).
UL:
DL:
C:

&&w3 Precision of decision transformation
A: 1) don't worry about it, the mapping will be close, use whatever line numbers is available.
2)
D:
J:
UL:
DL:
C: How mush of a problem is this?
Since no decision was made initially, this should remain on the agenda.

&&w4 new code added by transformation rule
A: 1) don't add any decisions
2) don't add any but notify the user (keep on the agenda)
3) add from the transformation rule if accidently difference handled by the rule
4) add decision from the union of the AST of the old
5) same as 4 but only if one view
D: A3 maybe - this is a conditional - need further study
J:
UL:
DL:
C:

&&w5 transformation rule decision views
A: 1) don't need to, there aren't that many
2) use DBSD as normally
3) Cross link to transformed system
D:
J:
UL:
DL:
C:

&&w6 manual after transformation
A: 1) Use DBSD to record any decisions
D:
J:
UL:
DL:
C:

&&

```
&&r5 tool_for_DHC
  A1: use Emacs in files .dd, .pd, .al(agenda list)
  A2: create a dedicated editor

  D: A1)
  J: programming ease
  UL:
  DL:
  C:

&&r6 robustness_of_DHC
  A1: Programming ease.
  A2: Robusteness.

  D: A1
  JUL: tool_for_DHC
  UL:
  DL:
  C:

&&r7 order_of_objects_editing
  A1: write our own interpretor
  A2: check consistency every time we use a DHC command
  A3: check consistency on exit
  A4: rely on user to be "nice".

  D: A3
  J: In dhc_quit we check the .dd and .pd files for consistency of the objects.
     Check every field and modify in the Agenda Status.
  UL:
  DL:
  C:

&&r8 insert_agenda_list
  A:
  D: For each problem, put the following entries:
     &&<problem-description-name>:
     \tDate Entered: <the time when the probelm was entered> ;filled automatically
     \tDue Date: <the time when the problem will be solved> ; blank initially
     \tResponsible Engineer: <the person responsible for solving this problem> ; in
                              the person who created the problem
     \tPriority: <an integer ranger from 0 to 100, 0 - lowest and 100 - highest>; b
     \tStatus:       ; Currently there is only the "Empty Fields" sub-entry
     \t\tEmpty Fields: <the blank entries of the problem in DHC.pd and DHC.dd> ; al
     \tComments:
  J: provide whatever is needed.
  UL: describe-problem
  UL: make-decision
  DL: terminal
  C:


&&r9 modify-agenda-list
  A1: modify automatically.
  A2: let users do it.
  D: A2
  J: simple, and some entries such as "Priority" can not be set automatically.
  UL: insert-agenda-list
  DL: terminal
  C:


&&r10 entry_definition
  A:
```

D: an entry starts with "&&"
J: For an easy identification of the entries.
UL:
DL:
C:

r11 field_definition
A:
D: An entry field begins with TAB@@.
J: For identification purposes.
UL:
DL:
C:

r12 empty_field_definition
A1: Just a CR.
A2: Without any character.
D: A1- need to be validated later.
J:
UL:
DL:
C:

&z1 logging-form-representation
A:1). a form of the process model and checkout the steps that I am doing.
2). a list of the terminal activities from the process model. In this case I loose the sequence of activities and the objects on which I do the activities.
D:
J:
C: The form we need should state the activities, the time spent on each of them, the order of activities.

&z6 CASE-tools
A:1)use DHC alone;
2)use Refinary alone;
3)use both DHC and Refinary;
4)Use other tools.
D:We choose the third alternative.
J:
C:

&z7 dhc-refinary-integration
A: 1) make them loosely coupled, seeing each other like a black box that executes its job sequential in time with respect to the other tool.
2) make them tightly coupled.
D:
J:
C:For a tightly coupled version we will need to embed DHC in Refinery, to consider each of the Refinery functions as black boxes and wrap them in DHC functions, if they are sufficiently small. Also we would need that this functions be noninteractive so that we can have the control of the user actions at the DHC level. For this we would need a deep understanding of the source of Refinery, to figure out how to make the link with DHC. We would need from Paramax a detailed list of capabilities and functions of Refinery.

&z13 decision-views
A: We do not wish to have several operations active at the same time; i.e., only when the code is completely ported to the target machine then is it turn over to the user (from porter's view to the user's view).
D:
J:

```
UL:
UD:
C:

&&
```

Preliminary Draft: 11/18/92

ATTACHMENT #6

Process model for traditional life cycle of software notations specific to DHC

Notations

--> - is defined as

Capital - a label for a subspace of the process model space - an intermediate symbol - a set of activities done often enough to merit its own name.

lower - name of an object either needed for an activity or produced by an activity

> - indicates data (object) flow (input or output)

$Capital_{subs}$ - subs is the person normally engaged in this activity

boldlower - functionality available in DHC or process model

[ ] - repeat 0 or 1 time

{ } - repeat 0 or more times

| - alternate paths in subspace

$activity_1 || activity_2$ - activities which can be done in parallel

$activity_1$ "blank" $activity_2$ - activities which are done in sequence

( ) - used for grouping, but does not assign an intermediate name to it.

/*comment*/ - can be used anywhere in process model to help explain

name:object - names an instant of an object in the process model

∇{objects} > Activity - perform activity for all objects in set in random order

Process Model Objects

Software = {lines of source code}

Documentation = (problem_space, decision_set, documents, justification_relation, dependency_relation, decision_relation, alternate_relation, task_relation, description_relation, view_relation)

Documents = (requirements, specification, design, source)

Alternate_relation = (problem x problem)

Justification_relation = (problem x problem)

View_relation = (problem x document)

Dependency_relation = (problem x problem)

Decision_relation = (problem x decision)

Task_relation = (problem x task_problem)

Description_relation = (problem x problem_description)

Works_with_relation = ((manager,task), (software_engineer, problem))

Problem_space = {problem}

Decision_set = {decision}

System = (software, documentation)

Environment = {DHC}

Schedule = {(task, problem, person, status, priority)}

Agenda = {(problem, progress information)}

Session = $documentation_{task,\ timeB} - documentation_{task,\ timeA}$

Problem_description = ((description, alternates, decision, justification), attributes: (abstraction_level, generic, user_filter, size, reuse_list)),

/* generic: a node in generic if it is contained in all alternatives of its father*/

/* user_filter: used to create user defined filter*/

/* notes: are created during the understanding and assessment phase*/

Size = (# problem, # LOC, # documentation, # filter (documentation))

Task_problem = (problem, (reuse_list, generic_list, out_list, modify_list, new_list)
risk, effort, lower bound: size, upper bound: size)}

/* task is a request from either the customer or the manager to change some aspect of the externally observable behavior of the system. Adaptive task - change a requirement; perfective task - change decision in a problem; corrective task - change software and/or documentation of a problem node whose solution is incorrect. It is represented by a problem_description*/

Persons = (MAnager, SOftware_engineer, CUstomer)

Report = {lines of text}

Meeting_notes = {lines of text}

Process Model for Traditional Life Cycle

Detailed Description

External_customer_requirement_resolution$_{CU}$ --> *task*$_{CU}$ > (Software_development
|| Customer_feedback) > system

Internal_perfection_and_correction$_{CU,MA}$ --> *task*$_{CU,MA}$ > Software_development >
system

Software_development --> task > (*Understanding*$_{MA}$
Understanding$_{SE}$) >/*requirements_definition*/ task_root: task_problem
/*list_of_reusables candidates*/ {problem},
Task_problem_solving > /*first_level_decomposition*/ {task_problem},

{*Assessment*$_{MA,SE}$ > (task_root, effort, task_root.size, task_root.risk) Change_task_decision > task_root}

Assign_resources > schedule

Transfer_task_to_problem_space > agenda

{agenda; schedule > *Solve_problem*$_{SE}$ > agenda
Review_meeting > meeting_notes
Implement_meeting_decision > schedule, agenda}

Understanding$_{MA,SE}$ --> Exploring || *add_to_report*$_{MA,SE}$ > report

Exploring --> Requirements_definition || Reusability_search

Requirements.definition --> (keywords > **locate_problem** > relevant_nodes: {problem},
make_new_requirement) > task_root_problem

∇ relevant_nodes > Understand_problem

Understand_problem --> problem >{(Visit_node **dependency_up** > problem)
| **terminate_at_node_closure_relevant_nodes**
| **back**}

/*exploration*/ (**justification_from** > problem
| **justification_to** > problem
| **dependency_up** > problem
| **dependency_down** > problem)

Visit_node --> problem > **read_description document_view** Read_document Read_justification

Read_document --> {(**switch_view** /* special view: file view */ | **problem** | **decision** |
back | **scroll_view** | **emacs_commands**)}

Read_justification --> ∇ {**justification_to** & **justification_from**} (**read_description** | Visit_node)

Task_problem_solving --> task_root > (∇ node ∈ relevant_nodes) (Modify_existing_node > {task_problem}
{task_root > Create_additional_new_task > task_problem})

Modify_existing_node --> node > **create_task_problem** > task_problem, task_relation
tag_subproblems Modify_problem_node

Modify_problem_node --> node, task_problem > {Add_new_feature | Delete_old_feature
| Change_old_feature | Copy_generic
| Add_reuse /* for all nodes on reuse list}

Add_new_feature --> node, task_problem > **change-description add_justification**
create_new_task_problem
add_to_new_list

Delete_old_feature --> **add_to_out_list**

Change_old_feature --> node, task_problem > **change_description adjust_justification**
create_new_task_problem > modify_node, modify_task_problem >
Modify_problem_node
/* stop decomposition of modification at point where it is possible to estimate size */
add_to_modify_list

Copy_generic --> **change_description adjust_justifications copy_new_generic_task_problem add_to_generic_list**

Add_reuse --> **change_description adjust_justification adjust_reuse_task_problem add_to_reuse_list**

Create_additional_new_tasks --> **create_new_task_problem**
First_level_decomposition > task_problem

First_level_decomposition --> task_problem > {**create_new-task_problem add_to_new_list**}

/* create decomposition problem nodes for each task problem node and tag (create a list of) them as *generic*: to be used in all alternative solutions, *out*: not to be used in the new task, *reuse*: to be used with minor modification, *new*: a new feature is to be added to the original solution, *modify*: one or more subproblems have to be added, deleted, or changed */

Assessment --> Calculate_Direct_Effect Calculate_Indirect_Cost Review_$Data_{MA}$

Calculate_Direct_Effect --> (∇ task ∈ {task_problem})(∇ task_node ∈ task.generic ∩ task.out ∩ task.reuse)
calculate_node_size calculate_risk_effort > {task_node}

(∇ task_node ∈ new) $effort_{SE}, risk_{SE}$ > **get_effort_risk_size_estimates** > {task_node}

(∇ task_node ∈ modify) Assessment > {task_node}

{task_node} > **calculate_task_problem_size** > task
/* for each category add the number in the subproblem nodes to obtain the relevant figures in the task problem node*/

{task_node} > **calculate_task_problem_effort** > task
/* this is the sum of the efforts in the subproblem list */

{task_node} > **calculate_task_problem_risk** > task
/* the sum of the risks in the subproblem nodes */

{task} > **add_up_direct_costs** > task_root

Calculate_Indirect_Cost --> relevant_node > **get_closure_list_justification_to_from** > ripple_list: {problem}

/* get the worst possible impact by calculating the transitive impact closure for the justification limits */

∇ (node ∈ ripple_list) **calculate_total_node_size** > {task_problem.upper_bound}

/* add up all the metrics, # problems, #LOCS, etc, for all the nodes in the closure */

/* allow for interactive estimates */

|(∇ node ∈ ripple_list) (**read_description** (**calculate_total_node.size** | **skip**)) > {task_problem_lower_bound}

/* add only selected nodes to the calculation */

{task_problem} > **add_up_indirect_cost** > task_root

Review_Data --> {({relevant_node}, {task_problem}, task_root, ripple_list) > Pick_node **read_description**}

Pick_node --> **dependency_up** | **dependency_down** | **justification_to** | **justification_from** | **task**

Reusability_search --> **add_reusable_node** > task_problem.reuse_list

/* during exploration when finding candidate for reuse, add to reuse list of relevant node which is first in up chain of node in questions */

Change_task_decision --> {task_problem > **delete_task_problem** > node
(node > Modify_existing_node | task_problem > Create_additional_new_task) > task_problem}

/* delete a task problem and all its descendants and replace it with an alternate solution to the relevant node in the problem space or with an altogether new task node*/

Create_additional_new_task --> task_root > Make_node > task_problem

Make_node --> node > **create_new_problem_node** > new_node > Add_info_to_node > new_node

Add_info_to_node --> (**fill_in_description** | **link_justification** | **make_decision** | **write_and_link_documentation**)

Assign_resources --> **produce_schedule**$_{MA}$ > schedule

/* it is left to the managing system used to derive schedule from assessment data*/

Transfer_task_problem_space_to_task_root --> task_root > **transfer_tentative_to_problem_space** > task:problem, agenda, visible_alternative

/* transfer all task problems and their sub problems to the problem space, removing existing problems, documentation, source code which are to be modified but saving them on an alternative list for possible reuse. All new problems are added to the problem space. All incomplete problem nodes are added to the agenda. */

Solve_problem --> agenda > **take_agenda_problem** > node:problem
Add_info_to_node
{Make_node > new_node Solve_problem}
Adjust_agenda > agenda, report

/* any incomplete node has to be added to agenda and those completed can be taken off (to be saved in report) */

|| **add_to_report** $_{SE,MA}$ > report
/* notes on activities are added to a report */
| **research_problems** $_{SE,MA}$ > report

/* research needed to solve problems or prepare for the review meeting are done according to whatever system is prescribed and results in knowledge how to proceed with the Solve_problem activities or in a report for a meeting. This includes preparing a print-out of differences in decision graph, source and documentation. */

Review_meeting --> (Review_reports {(Generate_problems | Make_decision)} Review_progress)
|| **add_to_notes** $_{SE,MA}$ > meeting_notes

Review_reports --> (**review_report** $_{MA,SE}$ | **review_agenda** $_{MA,SE}$ | **review_schedule** $_{MA,SE}$)

Generate_problems --> {**describe_problems** $_{SE}$ | **describe_alternatives** $_{SE}$ | **give_justifications** $_{SE}$}

Make_decisions --> {**unconditional_decisions** $_{SE,MA}$ | **conditional_decisions** $_{SE,MA}$}

Implement_meeting_decisions --> {Change_decision | Add_unconditional_decision | Add_conditional_decision | Add_problem}

Change_decision --> **locate_decision delete_problem** > node
/*node is the parent of the problem deleted */
Make_node > new_node
Add_info_to_node > new_node
Adjust_agenda > agenda

Add_unconditional-decision --> **locate_decision** Adjust_agenda > agenda Add_info_to_node

Add_conditional_decision --> **locate_decision** Add_info_to_node **get_parent_node** > node
Make_node > new_node
/* create problem node for instrumentation problem */
Add_info_to_node
Adjust_agenda > agenda

Add_problem --> **locate_parent** > node
/* find the place in the problem space where this problem should go */
Make_node > new_node
Add_info_to_node
Adjust_agenda > agenda

Adjust_agenda --> **add_agenda_problem** | **delete_agenda_problem** | **modify_agenda_problem**

Preliminary Draft: 11/18/92

ATTACHMENT #7

Logging form

Tool Used	Activity Performed	Features Used	Start Time	End Time	Product