CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

# Concurrent Knowledge Systems Engineering

John C. Kunz

**WORKING PAPER**
**Number 5**

July, 1989

**Stanford University**

**Copyright © 1989**
**by**
**Center for Integrated Facility Engineering**

**If you would like to contact the authors please write to:**

*c/o CIFE, Civil and Environmental Engineering Dept.,*
*Stanford University,*
*Terman Engineering Center*
*Mail Code: 4020*
*Stanford, CA 94305-4020*

# Concurrent Knowledge Systems Engineering

John C. Kunz

*"Maximum anxiety causes anxiety."* Bob Filman

ABSTRACT

To build a knowledge system, it is necessary to specify the purposes in building the system, the domain description, reasoning and decision criteria, user and system interfaces, and a set of case examples to use in testing. These issues are offered as the major issues in building knowledge systems. This paper argues that they should be developed concurrently so that each is understood at an equal level of maturity at each stage of the knowledge systems development process. The "maximum anxiety heuristic" is an organizing principle to direct the attention of developers to these issues: the development process should focus on one of these issues until some other appears to be less well defined, or until some other causes greater anxiety. This opportunistic development activity can be used throughout the software development process, from concept definition through maintenance. The benefit of using this heuristic is that all of the basic knowledge engineering issues will be addressed explicitly, and addressing one helps to clarify understanding of the others.

## 1. Methodology

Based on the results of cognitive and computer science, human problem-solving can involve analysis of heuristics, algorithmic procedures, hierarchical descriptions of systems, and pictures. Thus, we can expect that AI reasoning, representation, and interface methodologies will each serve useful yet limited, distinct, and mutually supportive roles in the systems development process. The goal of the maximum anxiety heuristic is to provide a concrete and effective method for guiding knowledge system (KS) development activity so that the most important perspectives on knowledge systems are developed concurrently with equal maturity.

This section introduces the five issues of building knowledge systems and presents simple examples of each. The next section discusses the relation of this methodology to other problem-solving strategies, and Section 3 presents a brief case example of the application of the methodology in the early system and software requirements phases of system development.

### 1.A. Identify the Purposes of the System

In building a knowledge system, one question to ask is "What, precisely, are the purposes of the system", both from the perspective of the user and from the perspective of the expert. Creating a short statement of purpose helps focus the design team on solving the problem, as distinguished from issues related to the problem, and a simple statement of purposes helps other interested people to understand the project purposes.

Examples of statements of purposes might be:

- *The purpose of the system is to diagnose presence of any of three pulmonary diseases by interpreting pulmonary function test data.*
- *The purposes of the system are to help development engineers to identify all single fault failure modes in a system, to characterize system functions when running with such faults, to develop procedures for diagnosing and repairing those faults; and to help operators to identify faults in an operating system;*
- *The purpose of the system is to identify all feasible plans for manufacturing a particular metal part, given a set of raw materials; description of the part features, dimensions and tolerances; machines which are available for part fabrication; and characteristics of those machines.*

A useful design goal is that intended users -- users with good general domain knowledge but only rudimentary skills in computer use -- should be able to sit in front of a screen and discover most problems the computer can analyze with only a few moments of training and some browsing. This kind of access appears to be possible if the program embodies models of the problem domain area and of the problem solving process to which the user can readily relate. In addition, appropriate graphical diagrams on the screen, a small set of fixed top-level menus, and help text all help with achieving this goal of providing access to the technology by motivated untrained users.

Often, operational and executive users will have somewhat different purposes in wanting to use a system. Operational users often need to analyze individual problems while executive users often want to analyze multiple related cases. In attempting to identify the general objectives of the user, the knowledge system developers should attempt to identify the project gestalt. What is the problem faced by the user, and how does the user of the proposed knowledge system currently solve the problem? How are simple cases now identified and handled by users? How do users identify and handle difficult cases? The purpose of understanding the way problems are currently solved is not to form the basis for an exact model of current behavior, but rather to identify the issues which must be considered and the answers to particular problems. In general, the goal of the problem-specification process is to identify the purposes of building the

system, rather than to identify in detail how the problem is now or could be solved.

The purposes are the specific aims of the project, and they are implemented as top-level capabilities of the knowledge system. Thus, purposes can be viewed as the actions which can be performed by the knowledge system as a whole.

The purposes specify the top-level features of the system from the user perspective. These features are distinct from any system benefits. The usual project goals are to achieve the benefits. Thus, the project purposes are distinct from the overall project goals. For example, the purposes above might all be in support of projects whose overall goal is to help users to analyze particular problems better.

## 1.B. Describe the Representation or Model of the Domain

Knowledge system designers must describe the domain in which the problem exits, or the context in which decisions to be made. In general, it is useful to identify generic concepts of the domain and their attributes, as described by experts to peers and to interested novices. In addition, it is necessary to identify specific instances of the generic concepts as they are found in any modeled systems of interest. Represented concepts might include components, subsystems and states of a system, and attributes might include both measurable parameters such as height and voltage, and they might include states such as whether or not a subsystem is operational.

Examples of concepts and specific instances in a modeled system description include:

- *Descriptions of generic and particular parts in a system, such as heat exchangers, pipes, valves, and measurement instruments; description of the attributes of these parts, such as their specified dimensions and tolerances, cost, power consumption, and connectivity to each other; and principles describing flow of heat and mass in the plant.*

- *Description of generic and particular concepts in a problem-analysis theory such as project management, including concepts such as projects included in a program, activities of each project, capital and consumable resources, functional definition of cost and value, and policies for resource assignment.*

Domain representation can be viewed as the nouns which are to be described in the knowledge system along with their associated adjectives. Thus, problem domains can be represented conveniently as units and slots using frame-based representation systems.

## 1.C. Specify the Reasoning to Analyze the Model

Systems have behavior. For each specific problem to be analyzed, the knowledge system developers must identify criteria by which problems are to be analyzed and decisions are to be made. It is often necessary to specify the way those criteria are to be applied, or the control of the analysis process. Some decision-making criteria will describe implementations of principles of a field, such as an algorithm for computing the shortest path through a network, and other criteria will be heuristic, such as a process for approaching a diagnostic problem.

Behavior might include procedures to find related systems, diagnose problems, and display aspects of the structure or function of a system.

Examples of reasoning activities a system is to perform might include:

- *Identify all likely single faults in a system;*
- *Identify all machines downstream of a particular machine;*
- *Schedule machine operations in a factory using a particular scheduling heuristic;*
- *Infer some aspects of the behavior of a system by analyzing its structure and function.*

The reasoning defines the actions which objects can perform. The reasoning processes will reference particular attributes of particular objects, for example to determine the status and features of objects. In addition, the reasoning process will make conclusions about objects, or change the values of particular attributes of objects. Reasoning within a problem domain can be represented conveniently as rules or as algorithms in a procedural language. The reasoning procedures can usefully be associated with the objects they modify.

Reasoning procedures are often context-dependent: particular algorithms and heuristics work in some circumstances and not in others. One of the strengths of knowledge systems is that they can express the context in which a reasoning procedure is assumed to apply. The representation of a system should explicitly include description of the states and conditions of a system, and the reasoning procedures should then be conditional so that they apply when the appropriate conditions are met. The premise of an If-Then rule provides a natural place to state the conditions in which a rule applies.

Reasoning procedures are most flexible when they are generic. Thus, it is best for rules and methods to refer to variables or patterns of data, rather than to specific attribute values or to specific objects. Specific objects are best described as frames within the representation of the problem. Specialized

utilities and editors can be built to display and modify attributes of object descriptions easily.

Representation of a domain and reasoning about the domain are related. A useful simple distinction between the two is that the representation includes facts which can be asserted directly into or retrieved directly from the model. Reasoning is the process of inferring values which are not explicitly represented within the model. Thus, specification of the reasoning process must describe both of the relations among facts which are complex enough that they must be determined through an inference procedure and a strategy for carrying out the inference procedure. Typically, inference is performed at the time that a fact is asserted in the model or that an inquiry is made regarding the values of some attribute in the model. Thus, reasoning knowledge is normally implemented as rules or algorithms in a programming language.

Some reasoning may be performed by a representation system. For example, frame inheritance specifies the way that attributes and their values are passed from one object to some related objects. In addition, some systems support particular constraint propagation algorithms automatically.

A useful development procedure is to start the knowledge system development process by creating the rules and methods which can be used to analyze a particular test case. Then the representation and reasoning can be generalized as much as possible to handle a broader set of cases. Thus, initially rules and methods will be associated with and will reference particular objects, particular attributes and particular attribute values. During the process of generalization of the reasoning, the references to particular objects, attributes and attribute values can all be generalized to variables as possible and appropriate. The initial reasoning procedure may be rather heuristic.

While attempting to generalize the reasoning, a valuable exercise is to ask "Why" a particular heuristic or reasoning process should apply. Increasingly general reasoning procedures can often be developed by attempting to elicit the principles which underlie analysis of simple test cases.

### 1.D. Create User and System Interfaces

Knowledge systems typically include interfaces to the user, to sources of data and to destinations for results. These input and output data often lie in other systems applications on other computers.

The purpose of the user interface is to communicate the questions of the system to the user, solicit and receive input from the user, and communicate results of its analysis. When appropriate, the user interface also presents the reasons that the knowledge system made a decision or the assumptions made in the system about the structure and function of an application area. The most effective interfaces exploit natural idioms of a domain, including commonly-used forms, graphical diagrams and graphs.

Examples of elements of user interfaces include:

- *A tree showing relations among concepts in a knowledge base, such as CLASS-SUBCLASS, PART-WHOLE, or DOWNSTREAM.*

- *A graphical layout of a system, such as an architectural drawing for a building, or a schematic of machines in a factory or parts in an electromechanical system. The layout can include animation to show flow of parts, material, or information.*

- *Graphical network showing successful rule invocation. Rule graphs can be very useful for helping developers to extend and debug their applications, but their fine level of detail often makes them of limited value for users.*

The system interface is not required for stand-alone applications. Often, however, knowledge systems obtain data from existing data bases, and they return results to other applications systems.

### 1.E. Create Procedures to Test Model Validity

Systems must be tested repeatedly during their development to assure their validity and to determine whether changes fix identified problems or create new problems. Test procedures should include criteria for selecting individual test cases and, ideally, a "gold standard" for judging the accuracy of the interpretation of those cases. It is then necessary to identify actual individual test cases, the desired system responses for each individual test case, and procedures for comparing the system output for those cases with the expected results of analyzing those cases.

Examples of useful sets of test cases include:

- *The simplest case which makes any sense;*

- *Simple extensions of the most simple case and the way they are to be interpreted;*

- *A test case which is relatively complex which prototype versions of the system will not be able to solve but which a second-generation production system should be able to be solve;*

- *100 test cases (i.e., a large representative sample), identified over a period of time, which represent a broad set of situations of interest to the developers and potential users.*

The most useful suite of test cases includes all of the kinds of cases listed above.

These sets of test cases can be rerun each time a change is made to the system. By reviewing results of analyzing these test cases, developers can determine whether desired performance enhancements were made and identify whether any unintended changes were introduced into the analysis process. A test support system can be built to compare actual with desired test results and to report discrepancies and changes since the test set was run previously.

The first test case, the simplest which makes any sense, is crucial for developing the prototype. A measure of success and completion is that the initial prototype system successfully can accept this simple test case and interpret it properly.

One of the risks of rapid prototyping is building a prototype which cannot be extended to handle important difficult cases. The second test case identifies a difficult case of interest, and the set of 100 test cases normally should include a number of additional difficult cases. The purpose of identifying difficult test cases at the outset is to attempt to focus early attention on the issues of extension. The initial prototype should not be designed to handle the difficult cases. It should either be designed to be extensible to handle difficult cases, or when it becomes clear that the initial prototype will not accommodate the difficult cases, plans should be made to discard the initial prototype design and to create a second prototype which addresses the design issues presented by the difficult case.

## 2. Relation to other Problem-Analysis Strategies

Polya describes a related method for analyzing mathematical problems. Polya's "How to Solve it" method has four steps [Polya]:

- Understand the problem by understanding what is unknown, what is given, and the conditions on the problem. (He recommends drawing a diagram.)
- Devise a plan to solve the problem. (Attempt to identify a related problem and modify the plan which solves the related problem.)
- Carry out the plan
- Review the problem solution; check its reasonableness. (Check its sensitivity to varying assumptions; attempt to use the result or the method for some other problem.)

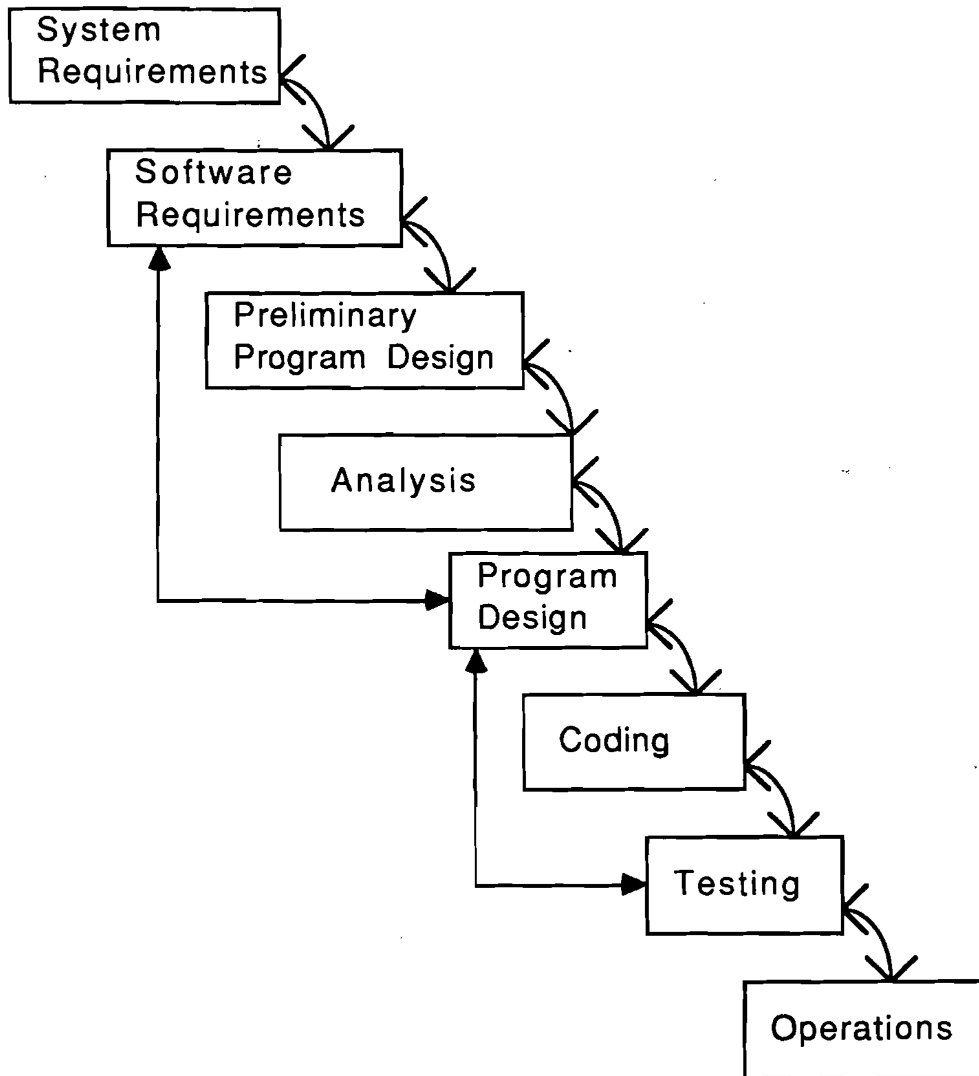
This four-step process was later elaborated and specialized in the software development waterfall, as discussed below. Polya's remarks about how to

perform these steps are what is particularly interesting about his method. He emphasizes using heuristics in problem solving, and he offers a number of useful ones. For example, he recommends using diagrams to describe a problem -- a technique widely used in classical physics, chemistry, engineering and now model-based reasoning [Kunz]. He recommends using a variant method of problem solving: he suggests identifying similar problems and modifying their solutions to solve the given problem. Finally, he recommends using sensitivity analysis to assess the robustness of the problem solution as input parameters are pushed to their expected extremes.

The five-issue maximum anxiety method is obviously close to Polya's method. Polya's first step relates to the "Purposes", "Representation" and "User Interface" issues in the building knowledge systems. His second and third steps relate to the "Reasoning" issue. His fourth step is a embodied in the maximum anxiety heuristic and analysis of each of the five issues of building knowledge systems. The five-step maximum anxiety method includes a number of heuristics to help guide the analysis, including all the major ones suggested by Polya.

An important difference between Polya's and the five-issue maximum anxiety method is that maximum anxiety emphasizes an opportunistic rather than an iterative control of the problem solving process. A second important difference is that Polya emphasizes use of a variant method: he suggests identifying similar problems and modifying their solution to solve the given problem. The variant method is most likely to be effective in circumstances, such as mathematics, in which there exists a well-understood body of existing problem-solution techniques to survey and to adapt. In knowledge systems, we are just beginning to develop a similar body of experience. Thus, while the variant method may become more effective in the future, we now see most developers building knowledge systems using ad hoc heuristic approaches or generating new methods based on analysis of basic principles.

The classic waterfall model of software development was described in 1970 by Royce [Royce] and has been elaborated often, such as in [Boehm, Davis]. The waterfall method was offered by Royce to describe and to prescribe the traditional software development process.




Waterfall software development process described in [Royce]

Most traditional software development methodologies have followed some variant of the waterfall method, often using slightly different names and slightly different numbers of steps.

Software projects are often delivered later and are more expensive than planned, and they often lack intended functionality and include bugs. One source of these software development difficulties is that requirements are difficult to specify precisely, and they often change as users see new possibilities and developers identify new ways to represent and analyze problems. Thus, while the waterfall method is simple to describe and easy to use in principle, it alone has not been sufficient to provide the basis for effective software development.

Knowledge systems technology has both helped and confused the software development issues. The hardware and software are more powerful then ever before. For example, in recent years the combination of interactive workstations and new software development environments have allowed development of knowledge-intensive applications which had not been attempted using traditional software technology, and they have allowed use of rapid prototyping throughout the development process. However, the KS problems are particularly difficult: if some problem is simple, it has been or will be solved using traditional technologies, so KS developers end with only the most difficult problems. Knowledge is difficult to specify when it is at all complex. Finally, the KS technologies provide powerful new techniques to work with, such as frame-based representation, object-oriented programming and production rules, but developers must learn to use these methodologies effectively.

The argument behind the five-issue maximum anxiety method is that it is valuable -- and with KS hardware and software it is now feasible -- to attempt to develop all of the knowledge systems issues concurrently as work proceeds down the software development waterfall. The five KS development issues should each be considered at every step down (and up) the waterfall. Royce argued that preliminary program design should consider program design, analysis, coding, testing and operations. (We now perform his preliminary design by rapid-prototyping.) His preliminary design step was valuable because it provided results for later use in program analysis, design, coding, testing and operation. The five-issue maximum anxiety method extends his argument and calls for attention to the entire development process throughout each step of the process.

## 3. Case Example

*This section is an edited transcript of a seminar discussion on Model-Based Reasoning in Engineering. Italics indicate remarks about the session; underlines indicate phrases written on the chalk board. LW is a Stanford civil engineering graduate student who volunteered to discuss his project. Prior to starting this knowledge engineering session, his research proposal stated that he plans to use "knowledge-based techniques in a deterministic approach that resembles simulation to produce precedence relationships for project activities [using] an algorithm which emphasizes a fundamental model of the planning process and exploits the use of project component relationships".*

JK: Let me get out a prop. Some of you know that I use an 'anxiety demon' *[An anxiety demon is a little finger puppet with long floppy arms.]* Tell us, Lloyd, about one of the five KS development issues, and let's talk about one of them, and when my anxiety goes up, I'll wave the anxiety demon, and we'll talk about the source of the anxiety, and then we will move on to address another issue.

LW: I think the most appropriate one to start with would be the purposes - or problem definition. I think the problem is to build a model, and since it is a model of construction planning, it should include the things in construction planning. It should be intuitive and within the usual operations of a person in the field who in fact is a scheduler. The purpose of the model is to build or construct construction schedules for projects.

JK: You said 'build model'. You said 'build schedules . . .'

LW: I'm being a little repetitive here. Build a model to construct a schedule in a way which is similar to the way that humans construct schedules.

*[The boxed items in this section show the description of the knowledge systems engineering issues as they were described on the chalk board during the discussion.]*

**Purposes: Build model of construction process**
- **Build schedules of construction projects**
- **Intuitive**
- **Natural to scheduler in field**

GL: Specifying how you are going to build it -- isn't that getting at the reasoning process?

JK: OK. What I think is going on is that Greg is suggesting looking at the problem from a different perspective. He is saying 'let's not talk about the purposes any longer; let's talk about the reasoning.' I think he just applied the maximum anxiety heuristic -- and that's good. In effect, he said: "I know enough about this issue for now, so let's go talk about some other issue." How are we going to solve this problem? Let's pursue this new issue.

HE: Is there a model of how people build schedules?

LW: I can only comment on how I do it. I usually start at the beginning of the project and work toward the end. Starting with the existing status of the project, which might be just an open field or a a level surface on a warehouse floor, and then I look at the set of plans and specifications -- primarily the plans -- and ask

what project components are there that need to be constructed, and given the current project status, what things do I need to do in order to complete the project.

> **Reasoning: Go from start to end of construction process**
> - **Look at current project status**
> - **Look at plans**
> - **Look at component specifications, identify components that need to be constructed**
> - **Identify things to do to complete project**

AB: So you have activities. What about durations of activities?

JK: I think we now have started to understand two issues, so as Anna has suggested, let's apply the anxiety heuristic and change the focus of the discussion from purposes or reasoning to something else.

GL: Anna just mentioned activities, which do not fit under the discussion of purposes or reasoning. That seems like the structure of the problem, or representation.

LW: Surely project components falls under representation.

Various: Activities. Status. Plans.

RL: The sense in which you said plans before was drawings.

JK: Here's an important observation. In engineering, we know that graphics are often important. . . . Let's look at the purposes and reasoning and see that we have a relatively complete list of nouns to include in the representation.

*[Schedules and projects appear to be basic concepts underlying the discussion so far, so while not mentioned explicitly during the discussion, they were added to the list of things to describe in the scheduling model.]*

**Representation -- Things to describe in the scheduling model:**

| | |
|---|---|
| **Project beginning** | **Project End** |
| **Project status** | **Level surfaces** |
| **Drawings** | **Specifications** |
| **Activity durations** | **Activities** |
| **Schedules** | **Projects** |
| **Components to construct** | |

LW: And then we are going to end up with a bar chart.

JH: By bar chart you mean Gantt chart? And probably some sort of precedence network?

LW: I'm not certain [about the need for a precedence diagram].

JK: We are discussing some user interface. Let's add bar charts to the user interface list.

*[When we discussed purposes, drawings and flow charts showing steps in the construction process appeared to be user interface items. They were listed under user interfaces on the chalk board to make these interface features explicit.]*

**User Interfaces:**

**Construction drawings**
**Bar charts (Gantt chart)**
**Flow chart of steps in construction process**

**System Interfaces:**

GL: Don't we have multiple models now? The bar chart is a model of the schedule, and we have a model of the building . . . That is, we have a model of the construction process with its activities, resources and time. The other is a model of the physical building, and that looks to be a different model. And we have to describe them both.

HE: At this point I'm still confused about the scope of the problem. What are the inputs that you want to provide, and what are the outputs you want to get?

JK: Hossam, what you did was to raise your anxiety demon. You said "I'm confused", and then you went back to the purposes. That's really good. You were looking at representation, and you said I don't know what the purposes of the system are.

In another session, we discussed an iterative versus an opportunistic control strategy. Here is an example of our applying opportunistic control. We are not starting at one place in a list of things to do and then going down that list. We are focusing our attention wherever we seem to have the most information to gain. I think that the opportunistic or anxiety-driven control strategy really helps the KS development process because it helps us to develop each of the issues with equal maturity.

Let's continue with anxiety. Notice that we have discussed four issues now. My anxiety demon is up now. Let's talk about testing.

LW: I would compare the results of this model with the results of a human scheduler.

RL: Would you compare the decision process along the way and compare intermediate steps, or would you look only at the end result?

LW: I would look at the beginning and end of the process, but not the intermediate steps.

JH: I would suggest doing the opposite. When you start the process, you want to see that your model is working efficiently and the way you think it ought to.

JK: Good question. How do you decide whether to look at the process early on or not? How do you make that decision? How do you ask that question?

HE: Doesn't it depend upon what the problem is . . .

JK: Exactly. You can't look at the particular issue of what to test in more and more detail and resolve the ambiguity. You can resolve the problem only by looking at one of the other KS development issues -- in this case, the purposes of the system.

HE: He said he wants to bring a model of the scheduling process into the system.

JK: Now, you can decide to actually model the decision-making process, and recognize and accept that a lot of work is involved, or you can say that the process is not really a crucial issue for your purposes, and you can finesse it. We can't say now how to resolve the issue, but we have pointed out the

consistency which should be maintained between the purposes, representation, reasoning and testing.

> **<u>Test procedure</u>**
>
> **Compare KS schedule with one built by a human**
> **Process: No**
> **Result: Yes**

LW: I would say that we should test the process, not the results. That's the opposite of what I had just said.

GL: So what you are saying is that you want to achieve the same scheduling results, but you also want to follow the same reasoning path as the human scheduler.

LW: Yes.

JK: One additional testing issue. It would be good to look, real soon, at what is the <u>simplest test case</u> that makes any sense at all. You should make the comparison, but you should identify a test case which would be recognizable by another professional as being real, but which is so simple that you can make some progress with it quickly. So, what is that simplest case?

DJ: You mean a case for a simple prototype?

JK: Yes.

LW: Here it is. *[LW puts a diagram of a simple house on the table].*

> **<u>Test procedure</u>**
>
> **Compare KS schedule for constructing a building with one built by human**
> **Process: ~~No~~ Yes**
> **Result: Yes**
> **Simplest possible test case: playhouse**

JK: OK, let's look where we are. We have been going for 10 minutes, and we have said a little bit about each of the KS development issues, and that is good.

HE: I'm still not sure about the inputs and outputs.

JK: We can now turn on our anxiety demons and ask what we are most uncertain about. Now is a good time to scan all five issues, anxiety demon in hand, and ask what we have greatest anxiety about, and we can explore that issue.

JH: A lot of people have anxiety about the purposes in building the system, and I think he wants to model the process of constructing a schedule, not model the construction process -- not model how you build the building and crane movements and so forth -- but model how the scheduler analyzes the scheduling process.

LW: I agree. I want to model the process of building the schedule, not the process of building the project.

GL: So the thing you are modeling is the human expert scheduler.

JK: You are modeling the process of scheduling.

Do we have anything in representation that talks about the scheduling process? I don't see it . . . So we should add something. We don't know what it means yet, but since the concept is key to your work, we will have some representation of it so that the concept description is explicit. We will have some reasoning in support of it. We will have to see it through the interface, test it, and of course we have some purpose in considering it.

How are you going to know a process when you see one? Is it a list of steps?

LW: It is an algorithm. It will have boxes and arrows. A flow chart.

HE: At this point, I would suggest looking at a small example -- the playhouse.

Various: Discussion about purposes and reasoning.

JK: Look at what we have said so far about each of the issues. People are asking about the purposes and the reasoning. So we are applying the anxiety demon. This questioning is good. My only counsel is to not spend lots of time deciding what to do next because we want to develop each of the KS issues with equal maturity, so we will address not one or another issue but both. The only question is the order of addressing issues.

Let's make an arbitrary choice. Let's look at the reasoning. Maybe you can give us a 30 second overview of how the reasoning takes place.

LW: I can't do it in 30 seconds. I can describe the whole algorithm I had come up with in three or four minutes. But, we can look at it at a high level.

So, I figure out where the project is going to be built. Let's assume that it is a warehouse floor where we need no special preparation. It is 16' x 16'.

JK: Wait now. Identify building site?

LW: Yes. It goes back to the current status comment under the description of reasoning. Hopefully nothing will have to be modified on the site.

So I look at the little building and I decide what to do first. Say I choose a stringer on the bottom which is 6 inches by 6 inches by 8 feet long. I go grab it, and I lay it down on the warehouse floor.

RL: How do you know you do that first?

LW: Based on the fundamental principle -- and this principle should be included explicitly in the representation -- that you have to have support for physical objects before you can use them.

JH: Haven't you skipped a step? Haven't you already broken the plan down into objects? One of your components is a stringer, and it is at the bottom . . .

JK: The focus of the discussion has changed -- from the playhouse as a whole to the physical components of the playhouse. So you made a transition change.

LW: That's not the only change. I changed from talking about a problem component to an activity. You gave me only 30 seconds . . . *[Laughter]*

**Reasoning: Go from start to end of construction process**
- **Look at current project status**
- **Look at drawings**
- **Look at specifications of components**
- **Identify things to do to complete project**
- **Use principles**
- **Break drawing into components**
- **Map components to activities**
- **Find activity precedence using support principle**
- **Find activity duration**

GL: So far you have told us about the building drawing, but you haven't said anything about how you reason about the drawing. You went directly from a building to a site to placing a stringer. We are talking about the results of the scheduling process, and not the process. Before, you said that the site did not need modification. How did you decide that it needed no modification?

JK: For now, let's ignore the assumption that the reasoning can be specified. We can do a left-to-right depth first search of the reasoning process and get bogged down in enormous detail which will not be important in the grand scheme of things.

LW: The next thing is to map components to activities. That is a complicated mapping. *[Discussion]*

JK: Let's pop up a little. This discussion of how to do the mapping from components to activities is good. I think what is going on in general is that we are focusing on what the system inputs are: in particular, is the object-activity mapping an input to the system, or does the system have to provide reasoning to perform the mapping? The details of the issue are important, but it is also important to know where the particular issue fits into the KS development process.

RL: We have very much a problem-definition issue.

JK: Right. The functionality of mapping is required; we have definitely agreed on that. Either mapping is input, or it is to be provided within the proposed system, or more likely, there will be some of each with some mapping input and some reasoning to elaborate and apply the input mapping.

LW: OK. I propose that the inputs include a list of activities and their associated objects for the proposed building. Also, include the relations among those objects, such as topological -- supported-by, enclosed-by and adjacent-to, for instance.

**Representation:**

| | |
|---|---|
| Schedules | Projects |
| Project beginning | Project End |
| Project status | Level surfaces |
| Drawings | Building Sites |
| Components | Activities |
| Buildings | Site preparation |
| Floors | |

Scheduling Process (Flow charts)
Specifications: (lengths, widths, depths)
Relations: (enclosed-by, adjacent-to, Supported-by)
Actions: (Choose, Grab, install components)
Principles: (object needs support before installation)

GL: Now are these inputs that the end user provides?

JH: Whether it comes from a user or the keyboard or a file isn't important.

JK: We haven't talked about system interface yet, just user interface. Is it fair to say that there is no planned system interface: all input will come from the user using the mouse and keyboard and all output will be on the display screen?

LW: Yes.

RL: So the question is whether the list of activities is part of the permanent knowledge base, represented as objects or rules or something, or whether the activities are simply data for the program, to be input by a user?

JH: So, we are not focusing on going from a drawing to a schedule, which is a very large problem, but rather, given some activities and objects and some topological relations among objects, inferring the schedule. So this issue is a very small piece of the whole scheduling problem, so we can say that the system will not be terribly useful to Joe contractor down the road, but that's OK.

JK: So the system interface is a Nil? No CAD interface. No project management software. No finite element analysis. The system will be self-contained.

LW: Yes.

**<u>Interfaces</u>**
**User Interfaces:**
**Construction drawings**
**Bar charts (Gantt chart)**
**Flow chart of steps in construction process**

**System Interfaces: Nil**

GL: The system does not reason that to build the building takes particular activities. That reasoning is done by the user before using the system. So, back to the purposes, we assume that somebody has already identified the activities, relations and precedences.

LW: No, the system identifies the precedences. The <u>system output is the precedences</u>.

RL: The output is the precedences <u>plus the activity durations</u>. The distinction we make between planning and scheduling is that planning creates the list of actions. Scheduling derives their sequences and the times they are to be performed.

**<u>Purposes</u>: Build model of construction scheduling process**
- **Construct schedule of a construction project**
- **Intuitive**
- **Natural to scheduler in field**
- **Inputs: Activities, associated objects, relations**
- **Outputs: Activity precedence, durations**

GL: I have a question about representation. If the activities and their relations are input explicitly, why do we need to represent drawings? What do we need those drawings for if we are not going to reason about them?

JK: We did something with respect to maximum anxiety. What did we do? We looked at what had been written before and critiqued it and found questions

about it. Greg applied the anxiety demon to what had been done, while before we had applied it to what had not yet been done. That's good.

GL: We had assumed that the system would take those drawings and produce a schedule from those drawings. That's not what we are doing now.

JK: We have spent a half an hour now designing our system. This is a good time to go back and critique what we have done. So is the representation a good one? Again, we have to ask that question by looking at the various KS development issues and making sure that our representation describes the things we need to achieve our purposes, look at the way the reasoning is supported by the representation, and so on.

RL: These notions of objects and activities and relations: we haven't discussed how to implement them as objects or relations or attributes or rules or whatever.

JK: Right. We have spoken so far only informally about engineering concepts. Let's assume that we are now close enough on our basic engineering approach to the five KS development issues, and let's start to formalize these descriptions using the specific idioms of our knowledge systems development environment. With respect to the software development waterfall model, we are now going to jump from the system requirements to the software specification step.

Representation is a good place to start with formalizing the engineering notions which we have described informally. We want to end up with a representation of the generic concepts in our models of buildings, of drawings and schedules, and of the scheduling process. We can now look at our list of engineering notions under our representation list, identify the concepts or nouns, and identify the attributes of those concepts. I think the generic concepts might be listed as shown below. Each of these concepts represents a noun in the model of the scheduling process or the building model. Each of these objects has attributes, where the attributes of concepts which we have discussed so far are enclosed in parentheses.

**Schedules (Activities)**
**Projects (Schedule, beginning time, end time, status, components)**
**Level surfaces [Subclass of Sites]**
**Drawings**
**Sites (length, width, Level?)**
**Components (Components, length, width, depth, enclosed-by, adjacent-to, supported-by)**
**Activities (Object, start time, end time, successors, predecessors)**
**Floors [Subclass of Sites]**
**Footings and stringers [Subclasses of Components]**
**Principles (Defining rule)**
**Scheduling Process (Starting activity)**
**Specifications: (lengths, widths, depths)**

**Actions (Object)**
**Grab [Subclass of Actions]**
**Install [Subclass of Actions]**
**Choose [Subclass of Actions]**

In general, each of the generic objects identified above may have associated instances. Many of the instances will be able to perform actions -- as specified in the reasoning. We could now describe these concepts as units and slots in a frame-based representation system.

One example reasoning procedure is the process which defines the principle that an object needs support before installation. In this case, the components might be able to determine whether they have the vertical support which is required before they can be installed.

Let's look in detail at reasoning which defines the support principle. The first rule shown below states that a firmly supported object is vertically supported and, from the perspective of support, ready for installation. The first premise limits the scope of the variable ?Object to all components defined in the KS. The second premise selects objects which have firm support, such as a foundation of a building assuming that the representation describes foundations as having firm vertical support. The conclusion states that the the value Vertically-Supported will be given to the Support-Status attribute of objects which satisfy the premises.

**If (?Object is in class Components) and**
**(The Vertical-Support of ?Object is Firm)**
**Then**
**(The Support-Status of ?Object is Vertically-Supported)**

The next rule below states that an object is also vertically-supported when its vertical support is Vertically Supported.

**If (?Object is in class Components) and**
**(The Vertical-Support of ?Object is ?Support) and**
**(The Support-Status of ?Support is Vertically-Supported)**
**Then**
**(The Support-Status of ?Object is Vertically-Supported)**

These two rules together define the transitive closure of the Vertically-Supported relation.

Since these two rules make conclusions about an attribute of components, in an object-oriented representation they will be associated with the generic components object. Together, they implement the principle that an object requires vertical support before it can be installed. These two rules could be

implemented using the actual syntax of a knowledge engineering development shell.

In a similar fashion, rules and procedural code could be written to implement the KS purposes and detailed reasoning capabilities as listed below:

**Find-Precedences (Project)** "Method to invoke rules and methods which construct a schedule of this construction project. This find-precedence process uses defined activities with their associated components and their relations."
**Find-Duration! (Activity)** "Method to invoke rules and methods which determine the duration of this activity. Note that determination of precedence and duration are independent processes."
**Show-Gantt (Activity)** "Method to invoke rules and methods which display a Gantt chart for the project, starting at the named activity. As necessary, this procedure invokes the Find-Precedences method to determine precedence of activities for the project, and it may invoke the Find-Duration method."
**Show-Drawing (Project)** "Method to display the graphical diagrams for this project."
**Show-Processes (Activity)** "Method to display the successor construction activities for this activity in a node-and-arrow tree form."

## 4. Discussion

An implication of these perspectives being distinct and mutually supportive is that the criteria for judging the effectiveness of work in one area is not within itself but rather with respect to its contribution to the other development areas. For example, the effectiveness of a set of frames cannot be judged by examining the frames themselves in ever greater detail. Rather, their effectiveness must be judged by their support for other knowledge system development issues: identifying purposes of the system, the reasoning to analyze the problem, and explanation of the problem-solution process. Similarly, rules and explanation facilities must be evaluated in the context of all of the issues of knowledge system development and testing.

Following the maximum anxiety heuristic is particularly valuable because some aspects of knowledge system development are relatively easier than others, and this heuristic forces developers to consider the easy and the difficult parts of system development concurrently. For example, representation of structural knowledge is usually relatively easy, whether done using a frame or a rule system. Thus, there is a natural tendency to focus on description of the domain and to avoid specification of complicated problem analysis. An even more insidious problem is that modern AI technologies support representation, reasoning and graphical interfaces relatively well, but they provide no direct support for problem specification or testing. Thus, there is a strong tendency to

build domain models, make them do something later, and finally figure out what they should be doing and whether or not they are doing it.

The methodology of rapid prototyping supports problem definition and testing. It allows developers to create precise problem statements and to demonstrate and test problem solutions quickly. A recommended technique is to propose a system development -- in a few written pages, possibly with some initial computer experiments to probe the feasibility of some basic technology; then to create and analyze a prototype system; then to create a functional system specification, and finally to start system development. The functional specification will evolve during system development, eventually becoming part of the user documentation.

In creating an initial proposal or prototype of a knowledge system, it is usually most useful to start addressing the knowledge system development issues by working on a whiteboard for an hour or so. Once an initial definition of each is in place, the developers can move to the computer and start creating elements of a system including units, rules, methods and graphical explanations. After some plateau has been reached with a system development, it is usually valuable to return to the whiteboard to continue applying the maximum anxiety heuristic with the five knowledge system development issues.

In starting building a knowledge system, a developer can initially focus on any of these development issues. Continual use of the maximum anxiety heuristic is important, but choice of issue to address initially is not important. During the first day or so of prototype development, it is appropriate to switch between the five development issues rapidly -- every few minutes. After several days of prototype development, it may be productive to spend up to an hour or so on any one development issue. Only after a functional specification is written and a development project is well underway is it likely that a complete day can be spent effectively on one issue without serious regard for other issues.

The sociology of the user environment usually affects all of the issues of knowledge system development. For example, the purposes of the system, the explanation and testing will each be affected by the environmental requirements for interactive or automatic operations, real-time or batch performance, and desirability of particular tabular or graphical input and output forms. Thus, developers must determine how a computer might fit into the working routine of potential users. While the expert must provide the requisite expertise for the system, it is important that developers learn the way users will function with a computer-based system and introduce user-oriented interface with appropriate editors, displays and reports.

Soliciting effective user input is a difficult and important process. Royce argues briefly for customer participation in the software development process, especially during requirements generation, preliminary system review, design review and in evaluation of test results. The QFD method [xxx] has been used to solicit user input regarding system design. The user input will be important in specifying the approach and much of the content for each of the five KS

development issues. Thus, the QFD and maximum anxiety methods are complementary.

The user objectives will help to specify both the nature of the user interface and the system explanation. In addition, the users' working environment will determine whether the system can be interactive or whether it must be fully automatic, whether a workstation is an appropriate delivery device or whether some other computer is more appropriate. In addition, if the current user environment includes use of a computer, it may be necessary to access some of the data used in the current system or some of its analysis results. Finally, it may be desirable to duplicate or at least to be consistent with its style of computer-user interaction.

The development process should include phased development with frequent milestones, heavy emphasis on testing, and regular deliverables. Usually, creating an initial testable prototype within one month is a realistic and useful goal. It is usually possible and appropriate to plan to place an application in controlled field test (alpha-test) in 6-12 months, in broader field test (beta-test) at 9-15 months and into field use within 6 months of starting beta test. The initial fielded system will be a useful subset of the eventual system, but there is usually so much benefit to early solicitation of user support that aggressive delivery is worthwhile.

This methodology has been used repeatedly and effectively in creating many different prototype applications. Some of these prototypes have been created starting with little more than some intuitions, and others have been created after a functional specification has been developed. The maximum anxiety methodology is particularly useful in specifying the purposes of the system, an issue addressed in some level of detail by creation of a functional specification, but it has also proven to be helpful in converting the statements of a functional specification into the computational idioms of the knowledge system development environment in which the project is being built.

The maximum anxiety heuristic continues to apply to the problem of creating a real application after a prototype has been built because the effectiveness of representation, reasoning strategies and explanatory facilities must be assessed by their support for other knowledge system development issues.

## 5. Summary

Throughout the knowledge system development process, opportunistically address each of the five issues identified below in bold face. The associated questions are useful in clarifying these issues.

**Purposes**

Why is the knowledge system being built?

Who are the users?

What work can the system do to help users work effectively?

What work can users do to help the system perform?

What are the inputs and outputs of the system?

How do representation, reasoning and interfaces generalize to related problems?

**Representation (Expressed as Objects with attributes and values)**

What are the specific things (nouns or objects) referenced in problem examples?

What are generalizations of specific objects?

What are generic concepts (Nouns or objects) of the domain?

What are measurable attributes of concepts and specific instances?

What are interesting but unmeasurable attributes of objects and instances?

What attributes are referenced by reasoning procedures?

What are relations among objects?

What are principles of analysis procedures?

Are all the objects and attributes used in reasoning or user interface?

**Reasoning (expressed as rules and algorithms)**

What are the conditions or constraints under which attribute values hold?

Are constraints described explicitly in the representation, or in rules and code?

What are the specific procedures used to analyze the problem?

In what contexts do reasoning procedures apply?

What are generlalized versions of specific analysis procedures?

What are the behaviorsof specific instance objects?

How can reasoning principles be specified?

How are behaviors propagated between objects?

**Interfaces (natural idioms, systems interfaces)**

What interface is required for the expected level of user sophistication?

What are the graphical idioms used by domain professionals?

What are tabular or text idioms used by domain professionals?

What are useful connections to other computers and systems to input/output data?

**Test procedures:**

What is the simplest possible test case?

How does (will) system output compare with expert analysis of the simplest case?

What are 100 representative test cases?

How does (will) the system compare with accepted interpretation of all test cases?

How are results explained in terms of principles of the analysis procedure?

Are results robust as input and state variables are pushed to their extremes?

## Bibliography

Boehm, B.W., "Software Engineering", IEEE Transactions on Computing, vol. C-25, pp. 1226-1241, Dec. 1976.

Davis, A.M., Bersoff, E.H., Comer, E.R., "A Strategy for Comparing Alternative Software Development Life Cycle Models", *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, October 1988, pp. 1453-1461.

Kunz, J., Stelzner, M.J., Williams, M.D.: "From Classic Expert Systems to Models: Introduction to a Methodology for Building Model-Based Systems", in **Topics in Expert Systems Design** (in press), North-Holland.

Polya, G., **How to Solve It,** Princeton University Press, 1945.

Royce, W.W., "Managing the Development of Large Software Systems: Concepts and Techniques", in Proceedings of WESCON, August 1970, pp. A/1-1 - A/1-9.

Acknowledgements: I appreciate the cooperation of the participants in Stanford University Department of Civil Engineering class CE217 for their thoughtful comments in creating the case example used in this paper.