

# Software Project Management Net: A New Methodology on Software Management

Carl K. Chang, Chikuang Chao, Think T. Nguyen

Mark Christensen

Department of Electrical Engineering and Computer Science  
The University of Illinois at Chicago  
E-mail: {ckchang,ttnguyen}@eecs.uic.edu

Electronics Systems  
Northrop Grumman  
Rolling Meadows, IL 60008 USA

## Abstract

*Managing the development of large-scale software systems is a challenge to all software project managers due to the ever-increasing complexity inherent in the software development life cycle. In this paper, a formalism intended to capture the concurrent and iterative nature of software development, called Software Project Management Net (SPMNet), is proposed to model software development projects. By augmenting and applying genetic algorithms in an innovative manner by sharply reducing the solution-search space complexity, our SPMNet provides optimal or near-optimal solutions to the resource allocation and project scheduling problems. Results collected from simulation runs clearly support our approach for practical applications.*

## 1. Introduction

There is a consensus that developing complex and high-quality software in a productive and cost-efficient way is a trend in the software industry for the 90's [1]. From managerial perspective, producing a large software system is full of problems inherent in any highly labor-intensive activity. For instance, a large work force must be assembled and organized into teams. The engineering and management processes have to be solidified. Requirements need to be specified to faithfully reflect customers' expectations. Plans have to be developed, and budget and schedules need to be mapped out and codified. Staff must be acquired, trained, and motivated to perform clearly delineated tasks. Resource conflicts need to be resolved. The entire project needs to be monitored and controlled continuously. Therefore, the challenge for software managers is to coordinate and control all the available resources so as to implement software to meet customers' satisfaction within time constraints with minimal cost.

In this paper, a new approach, Software Project Management Net (SPMNet), is proposed as the fundamental model for software management. In addition to the regular project management features, a set of advanced software management features can be derived based on SPMNet such as automating resource allocation

and scheduling based on genetic algorithms, generating structured activity network, and predicting a project's future status. The basic concepts and approaches for generating these features are basically divided into three areas:

- Automatic construction of structured activity network: Key concepts in this area include project planning spiral model [3], multiple levels of abstraction over the software development activities, and hierarchical Work Breakdown Structure (WBS).
- Intelligent Resource Allocation and Scheduling: Key concepts in this area include resource allocation, conflicts, optimal solution, scheduling, genetic algorithm, resource and complexity constraints.
- Prediction of future status of a Project: Due to space limitation, software project prediction will be presented in another paper centered on the issue of risk management.

This paper is organized as follows: Section 2 introduces the background and current status in software project management; Section 3 presents the formal definitions of SPMNets; Section 4 briefly discusses our intelligent approach for resource allocation and scheduling based on genetic algorithms; and finally Section 5 concludes this paper.

## 2. Software Project Management

Software project management is, like many other activities in the software process, a problem-solving issue. It involves what is to be done, a decision regarding how to do it, the control of how it is being done, and an evaluation (or measurement) of what was done.

The issue on "what" typically takes the form of a plan. Tausworthe [13] introduced WBS into software project planning in the early 80's. WBS provides a hierarchical view for the whole project, but the precedence relationships among the work packages are not clearly identified in the WBS. Currently, most of the project planning techniques used today are based on network-based techniques [7] [9], such as PERT and CPM, which was originated in the early 50's. However, the classical project management models are inadequate for large-scale distributed software project management,

since they are poor in modeling and analysis of the concurrent and evolutionary characteristics embedded in a software project [2] [6]. Liu et al. proposed DesignNet [6] as a formal method to describe the behavior of software development. However, considerations on resource allocation and time-related information are not included in the DesignNet.

The issue on “how” is the allocation of resources (e.g. a schedule or budget). Unfortunately, according to the survey in [14], resource allocation still heavily relies on software managers. That is, software managers need to manually assign the resources to different tasks based on their experience in order to optimize resource usage, shorten the cycle time, and control the evolutionary nature of project development. This is extremely difficult for large-scale software projects, which involve hundreds of tasks, programmers, and a wide variety of hardware and software resources. Since the search space for such a problem class is typically huge, even with computer tools we will not be able to find optimal solutions reasonably. In order to solve this problem, we propose a new approach to generate near-optimal resource allocation and scheduling based on genetic algorithms. We include a brief introduction of the algorithm in section 4. The detail description is included in [5].

## 2.1 Current Status of Project Management Tools

According to one survey of current commercial project management tools [14], there are over 150 tools available with different functionality on various platforms. In general, the functionality provided by these tools include project scheduling, resource management, project tracking, and project reporting.

The general project management process for all these tools can be summarized as shown below:

1. Defining the project Work Breakdown Structure.
2. Performing precedence analysis.
3. Assigning starting date and task duration for each activity.
4. Identifying and defining resources.
5. Making resource assignments manually.
6. Resolving resource conflicts (e.g. over-allocation of resources).
7. Gaining approval of the project plan.
8. Establishing project baselines.
9. Measuring and recording progress.
10. Making adjustments to the project plan.
11. Reporting project information.

Notice that project scheduling and resource allocation are treated as two separate problems in the above approach. In addition, the project manager handles the resource allocation manually. This approach is tedious and inadequate for large-scale software projects. In this paper, a novel approach using genetic algorithms, which combines project scheduling and resource allocation

together, is introduced to generate near-optimal project schedule and allocate resource automatically.

## 3. Software Project Management Net

Developing a large-scale and complex software system is a complicated job for software project managers, since they have to deal with the increasing complexity embedded in the software life cycle. There has been an increasing concern about the lack of an adequate formal model for managing the development of large-scale software system [1] [6] [10]. We introduce a formal software management model, SPMNet, as a theoretical foundation to address a variety of issues in software project management. One of the major advantages of this model is that it provides a formal communication method between customers, software managers, and software developers. Based on this formal model, the customers and the developers will be able to work together to monitor and examine the progress of the software project. Another advantage of SPMNet is that it provides the software developers a foundation to build tools that will support and enhance the software process. Furthermore, a set of advanced features such as visualizing the progress of a software project, constructing a structural project plan, automatic resource allocation and scheduling, and predicting the future status of a project can be derived based on SPMNets.

### 3.1 Introduction to SPMNET

There is a consensus that a single model to completely capture all facets of a software development process may not exist [1] [4]. However, we would like to extract a list of features that we feel are essential for successful software project management.

- The model should be able to support a variety of management functionality such as project planning, project scheduling, resource allocation, project tracking, project reporting, and project predicting.
- The model should be able to reflect the fact that software development is a design process. The design process is evolutionary in nature in which tasks will be constantly revisited, some tasks will be removed, and new tasks will be inserted.
- The model should be able to describe the parallel and concurrent nature inherent in software development.
- The model should be able to support abstraction by hiding the unnecessary details to reduce complexity so as to provide a clean high-level view.
- The model should be able to describe different kinds of activities and products at various stages of software development.

In order to capture concurrency in the software development process, SPMNets borrow some concepts from Petri nets [8]. SPMNet closely follows the

terminology used in Petri nets. However, the firing rules and information carried by tokens are different from traditional Petri nets.

A SPMNet consists of a set of places, a set of transitions, a set of constraints, and a set of arcs. There are four different place types including abstract activity, atomic activity, product, and decision. An abstract activity is a collection of atomic or abstract activities. By grouping several related lower-level atomic or abstract activities together, abstract activity provides a high-level view to software managers by hiding the underlying details in large-scale software projects. An example of SPMNet is shown in figure 1.

Each activity is associated with a set of constraints, which specifies the requirements for completing this activity. The constraints imposed upon an abstract activity are accumulated from its lower-level activities. The constraints can be further classified as resource constraints and complexity constraints. Resource constraints specify what kinds of resources are required and complexity constraints describe how much effort is needed for that activity. The software manager provides the constraints in each activity. Based on these constraints, genetic algorithms are used to determine the resource allocation for each activity automatically. Once the resource allocation is decided, the execution time for each activity can be calculated according to the complexity constraint on that activity. Software managers may pre-execute the SPMNet to visualize the project progress in advance. The condition place represents the success or fail decision after finishing an activity. The decision place is used to represent the iterative nature of the software development.

The dependency relations among activities are linked by transitions, product places and decision places. Note that there are three types of transitions,  $T_I$ ,  $T_O$  and  $T_{DO}$ .  $T_I$  denotes the input transition of an activity.  $T_O$  represents the output transition of an activity.  $T_{DO}$  represents the output transition of a decision place. Each transition type has different meaning and firing rules. The formal definition of  $T_I$ ,  $T_O$  and  $T_{DO}$  will be given in the next section. Following the links among activities, the precedence relationships among all activities can be derived.

### 3.2 Formal Definition of SPMNET

**Definition 3.1:** A SPMNet graph  $S$  is defined as a four-tuple  $S = (P, T, C, E)$  where  $P = \{P_{ab}, P_{at}, P_p, P_d\}$ ,  $T = \{T_I, T_O, T_{DO}\}$ ,  $C = \{C_r, C_c\}$  and  $E = \{EI_I, EI_O, EO_I, EO_O, EDO_I, EDO_O, EDO_I\}$ .

**Definition 3.2:** A set of places  $P = \{P_{ab}, P_{at}, P_p, P_d\}$  where  $P_{ab}$  for abstract activity,  $P_{at}$  for atomic activity,  $P_p$  for software product and  $P_d$  for decision. Different symbols are used to represent different types of places. A circle represents atomic activity place. A double-circle is used for abstract activity. Product place is drawn as a rectangle. Decision place is represented as a diamond.

**Definition 3.3:** A set of constraints  $C = \{C_r, C_c\}$  is associated with each activity, where  $C_r$  denotes resource constraints and  $C_c$  represents complexity constraints. For representing human resources,  $C_r$  is in the form of ( $\{[Required Skill][Required Proficiency]\}$ , (Maximal Allowable Units)), and  $C_c$  is in the form of ( $[Amount][Unit]$ ). To simplify the graphical representation in SPMNet, no visual distinction is made between  $C_r$  and  $C_c$ . The constraint is drawn as an oval.

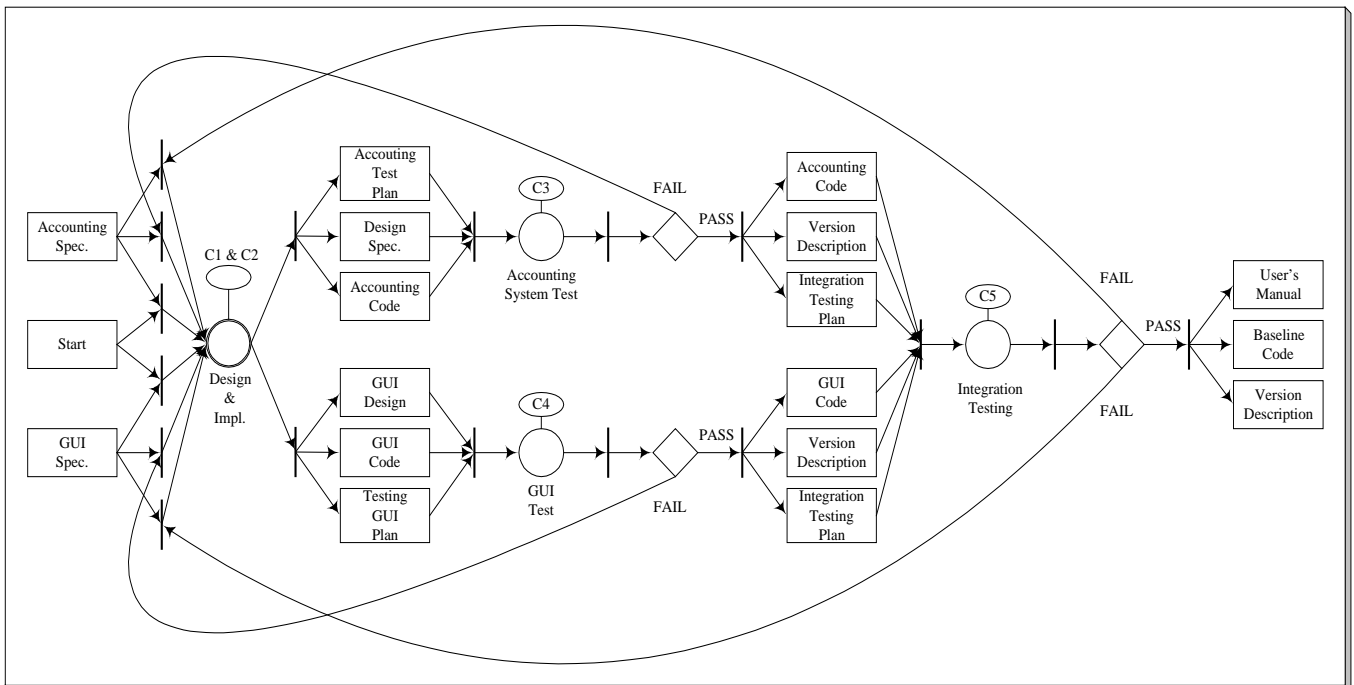


Figure 1. An example of SPMNet with abstract activity.

**Definition 3.4:** A set of transitions  $T = \{T_I, T_O, T_{DO}\}$ , where  $T_I$  is the set of input transitions to activity places,  $T_O$  is the set of output transitions from activity places and  $T_{DO}$  is the set of output transitions from decision places. In SPMNet, a bar is used to represent a transition.

**Definition 3.5:** A set of arcs, and  $E = \{EI_I, EI_O, EO_I, EO_O, EDO_I, EDO_O, EDO_I\}$ , where

- $EI_I$  represents the set of input arcs to transitions  $T_I$ , from product places.
- $EI_O$  represents the set of output arcs from transitions  $T_I$ , to activity places.
- $EO_I$  represents the input arcs to transitions  $T_O$ , from activity places.
- $EO_O$  represents the set of output arcs from transitions  $T_O$ , to product places or decision places.
- $EDO_I$  represents the set of input arcs to transitions  $T_{DO}$ , from decision places.
- $EDO_O$  represents the set of output arcs from transitions  $T_{DO}$ , to product places.
- $EDO_I$  represents the set of input arcs to transitions  $T_I$ , from decision places.

A tracking mechanism is associated with each activity place to keep track of all the events throughout the development life cycle. The activity related information such as the starting or end date of an activity and resource allocation of an activity is recorded in the activity tracking mechanism throughout the execution of the SPMNet. For each product place, the information such as version control and who are responsible for this product is kept in the product tracking mechanism. Similarly, decision places also keep track of all the decision-related information.

Unlike ordinary Petri Net, tokens in SPMNet are composite objects and carry more information. Tokens keep the execution history during the development of a software project. In other words, tokens keep a sequence of visited places. Tokens also record all the time dependent information such as starting/end dates of an activity, creation date of a product and starting/end dates for a decision place.

Transition firing rules are different from the firing rules of ordinary Petri nets. A checking procedure is associated with each transition to determine the enabling condition and the number of tokens created in the output places. In addition, different types of transitions have their own firing rules as defined below.

**Definition 3.6:** SPMNet Transition Firing Rule ( $T_{DO}$ )

An output transition,  $t \in T_{DO}$ , of a decision place is enabled if and only if there is at least one token in the decision place. The output transition,  $t$ , can fire if and only if  $t$  is enabled.

**Definition 3.7:** SPMNet Transition Firing Rule ( $T_O$ )

An output transition,  $t \in T_O$ , is enabled if and only if each of its input places has at least one token in it. An output transition can fire if it is enabled. The number of

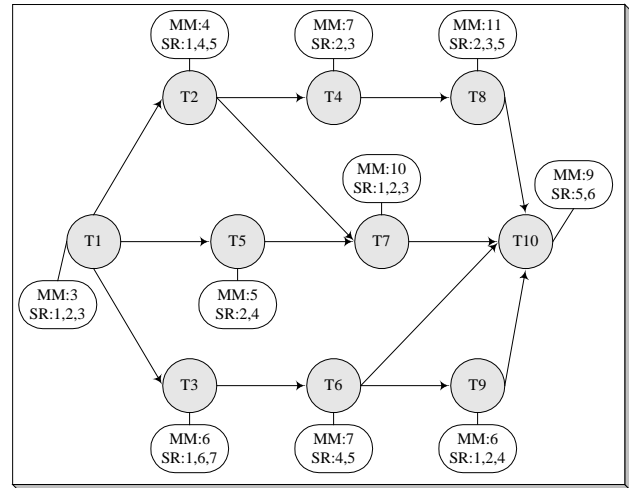
tokens generated in each product place is equal to the number of outgoing arcs in that product place.

**Definition 3.8:** SPMNet Transition Firing Rule ( $T_I$ )

An input transition,  $t \in T_I$ , of an activity place is enabled if and only if each input place of  $t$  has at least one token. An input transition,  $t \in T_I$ , can fire, if and only if  $t$  is enabled and  $t$  has not been fired by the token carrying the same product version information.

## 4. Resource Allocation and Project Scheduling

In the previous section, we propose the SPMNet as a fundamental model to represent a variety of activities in software development. SPMNet can facilitate a number of management functions such as project planning, tracking of the development history, automatic resource allocation and scheduling, and prediction of the future behavior of the project. In this section, we briefly introduce the mechanism for resource allocation and project scheduling offered by SPMNet. The details can be seen in [5].



**Figure 2. An example of TPG (MM: Man Month, SR: Skill Required).**

In our work, we explored the use of genetic algorithms (GA) in determining the allocation of software project resources. Genetic algorithms are used to simulate evolutionary mechanisms, and have been shown to be a robust solution-space search technique in a variety of optimization problem [11]. We apply the genetic algorithms to the problem of allocating staff so as to minimize the total cost and finishing time of a project.

The idea for scheduling and resource allocation is that a Task Precedence Graph (TPG) for each software project, including information on the estimated requirements of each task, is derived from a SPMNet. An example is illustrated in figure 2. The resource allocation schemes and schedules are calculated based on the TPG.

Using a database of employees and other resources, random sets of task allocations are created. This initial population of schedules evolves through the execution of genetic algorithms until an optimal or near-optimal match of resources to tasks is obtained.

In order to solve the resource allocation problem, a SPMNet is mapped to a directed acyclic Task Precedence Graph (TPG).  $TPG = (V, E)$  consists of a finite nonempty set of vertices  $V$  and a finite set of directed edges  $E$  connecting the vertices. The collection of vertices  $V = \{T_1, T_2, \dots, T_m\}$  represents a set of software tasks to be completed and each vertex consists of estimated effort (man-months or working days), required skills and proficiency of the skill. More specifically, for every,  $v \in V$ ,  $v = \{v_E, v_S\}$ , where  $v_E$  represents the estimated effort (man-months or working days), and  $v_S$  represents a list of skills along with the proficiency for each skill and maximal allowable units. The directed edge set  $E = \{e_{ij}\}$  ( $e_{ij}$  represents a directed edge from vertex  $T_i$  to  $T_j$ ) implies that a partial ordering or precedence relation (denoted by  $\rightarrow$ ) exists between tasks. That is, if  $T_i \rightarrow T_j$ , then  $T_i$  must be completed before  $T_j$  can be initiated.

#### 4.1 Knowledge-Augmented GA for Resource Allocation and Scheduling

Our approach is based on the augmentation of a genetic algorithm with domain-specific knowledge. In this paper, knowledge-augmented genetic algorithm is abbreviated as KAGA. The project schedule itself, instead of encoded bit-string, is used as a chromosome (also referred to as a string or a solution). Thus domain-specific knowledge can be used to design genetic operators which allow more efficient exploration of the search space around good points [12]. Domain-specific knowledge is incorporated into the schedule-generating, selection, crossover and mutation operations to ensure that they will

1. always yield legal schedules;
2. generate diverse and well-adaptive (fit) schedules;
3. guide genetic operators more directly toward better schedules.

Project	# of task	# of programmer	Combinations	Time (optimal)	Time (GA)
1	18	9	$1.07 \times 10^5$	1:01(min:sec)	3 sec
2	18	9	$2.92 \times 10^6$	28:48:00	7 sec
3	11	10	$3.75 \times 10^6$	25:08:00	11 sec
4	18	9	$6.81 \times 10^6$	65:58:00	8 sec
5	30	9	$4.73 \times 10^7$	674:40:00	12 sec
6	50	9	$1.00 \times 10^8$	2394:00:00	30 sec
7	10	19	$2.11 \times 10^8$	711:55:00	7 sec
8	18	10	$6.15 \times 10^8$	649:32:00	9 sec
9	15	19	$7.60 \times 10^{12}$	n/a	18 sec
10	20	9	$2.20 \times 10^{12}$	n/a	18 sec
11	18	19	$6.81 \times 10^{21}$	n/a	14 sec

**Table 1. Experimental results from exhaustive search and GA-Scheduling algorithm.**

By avoiding introducing unfit or poor schedules into the populations, this approach confines the search to the feasible and promising regions. Consequently, successful result and efficient performance can be achieved together. Basically, the algorithm uses small heuristically initialized populations, proportional reproduction (known as selection), one-point crossover, and single-assignment mutation schemes. In our problem, a legal schedule must satisfy the following constraints:

1. the precedence relation among tasks;
2. requirements in each task;
3. completeness of the schedule (i.e., all tasks must appear in the schedule).

The complete algorithm is shown in the following algorithm: GA-Scheduling.

#### **Algorithm GA-Scheduling**

**Input:** a task precedence graph,  $TPG = (V, E)$ . An employee database,  $D_{emp}$ . A skill database,  $D_{skill}$ .  
Parameter setting:

- POP: population set of schedule.
- NPOP: size of population.
- NEWPOP: new population.
- PROB<sub>C</sub>: crossover probability.
- PROB<sub>M</sub>: mutation probability.

**Output:** one near-optimal schedule,  $S_{refined}$

#### **Begin**

1. Generate initial population POP containing NPOP schedules.
2. Perform 3-operator Genetic Algorithm.
3. Do Step3-Step6 until converging; after converging, go to Step7.
4. Compute the fitness value of each schedule in POP.
5. Perform Selection.
6. Perform Crossover NPOP/2 times.
7. Randomly pick two schedules from NEWPOP and do the crossover with a probability PROB<sub>C</sub>.
8. Perform Mutation NPOP/2 times.
9. Perform Mutation for each schedule with an adaptive probability PROB<sub>M</sub> and put the schedule back in POP.
10. Refine the best-so-far schedule in POP.
11. Output the final near-optimal schedule.

$S_{near\_opt} \leftarrow S_{refined}$ .

**End;** {Algorithm GA-Scheduling}

#### 4.2 Experimental results

As a baseline for comparison of the performance of our domain-specific system, an optimal scheduling algorithm based on enumerative search was also implemented. The optimal scheduling algorithm, which requires exponential time to execute, generates all feasible combinations of schedules, and determines which combination generates the best fitness value. We implemented the programs using C++ on SUN-SPARC/10 workstation with a randomly set of experimental projects. Table 1 summarizes the

experimental results from exhaustive search and genetic algorithms. For the first eight projects, the results from the GA-Scheduling algorithm are identical to the results from exhaustive search. That is, the GA-Scheduling algorithm obtains the optimal solution.

## 5. Summary and Future Work

In this paper, we propose SPMNet as a formal model to facilitate the management activities during the software life cycle. SPMNet serves as a model to assist software managers in obtaining adequate understanding on the progress of a project for themselves and explaining to others the various steps that they must go through before completing a project. In other words, SPMNet provides a communication channel between the software developers and the customers and between the software developers themselves. However, there still are some limitations in our approach.

It is possible to assign the probability of success rate to each decision place in the SPMNet based on the software manager's experience. In this case, the transformation from SPMNet to TPG needs to be modified to represent the probability of success rate. As a result, the derived TPG includes cycles. The cycles inside TPG may be executed infinitely often unless the software manager specifies the allowable number of iterations and the related constraints for each iteration. It is possible to extend the cyclic TPG to an acyclic one (TPG') according to the information provided by the software manager. We can apply the GA-Scheduling algorithm to the extended TPG' to calculate the resource allocation and scheduling. However, it will be an expensive computation, since all possible cases need to be considered. In general, software management research is still in its early stage [9]. Lots of researches still need to be done in this area. Possible future research topics include:

- Incorporate additional factors such as risk management, software quality and reliability into the fitness function in the GA-Scheduling algorithm.
- Develop techniques for automatic configuration management system based on the SPMNet.
- Introduce visualization technique to SPMNet tools environment.

It is our belief that SPMNet provides a solid foundation to address a variety of issues in software management. Furthermore, we believe that SPMNet serves as a concrete model for building tools to support and enhance the software process.

## References

1. V. R. Basili and J. D. Musa. The future engineering of software: a management perspective. *Computer*, pages 90-96, Sep. 1991.

2. B. I. Blum. *Software Engineering a Holistic View*. Oxford University Press, New York, 1992.
3. B. W. Boehm. A spiral model of software development and enhancement. *Computer*, pages 61-72, May 1988.
4. B. W. Boehm and R. Ross. *Theory-W software project management: principles and examples*. *IEEE Trans. Software Engineering*, pages 902-916, Jul. 1989.
5. Chikuang Chao. *SPMNET: A New Methodology For Software Management*. Department of Electrical Engineering and Computer Science, University of Illinois at Chicago.
6. L. C. Liu and E. Horowitz. A formal model for software project management. *IEEE Trans. on Software Engineering*, 15(10): 1280-1293, Oct. 1989.
7. K. Lockyer. *Critical Path Analysis and other Project Network Techniques*. Pitman, London, 1984.
8. T. Murata, V.S. Subrahmanian, and T. Wakayama. A Petri net model for reasoning in the presence of inconsistency. *IEEE Trans. on Knowledge and Data Engineering*, 3(3): 281-292, Sep. 1991.
9. J.J. O'Brien. *Scheduling Handbook*. McGraw Hill, 1969.
10. D. J. Reifer. *Software Management*. IEEE Computer Society Press, CA, 1993.
11. M Srinivas and L. M. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6): 17-26, Jun. 1994.
12. J. Y. Suh and D. Van Gucht. Incorporating heuristic information into genetic search. *Proceedings of the Second International Conference on Genetic Algorithms*, pages 100-107, 1987.
13. R.C. Tausworthe. The work breakdown structure in software project management. *J. Systems and Software*, 81:181-186, 1980.
14. SCST Management Team. *Project Management Technology Report*. Software Technology Support Center, UT, 1993.