December 1993

PM-Net: A Software Management Representational Model

Kuen-Ching Lee  
National Sun Yet-sen University

Hsin-Hui Lin  
National Sun Yet-sen University

Iuan-Yuan Lu  
National Sun Yet-sen University

Follow this and additional works at: http://aisel.aisnet.org/pacis1993

Recommended Citation
http://aisel.aisnet.org/pacis1993/63

This material is brought to you by the Pacific Asia Conference on Information Systems (PACIS) at AIS Electronic Library (AISel). It has been accepted for inclusion in PACIS 1993 Proceedings by an authorized administrator of AIS Electronic Library (AISel). For more information, please contact elibrary@aisnet.org.
PM-Net: A Software Project Management Representation Model

Kueh-Ching Lee¹, Hsin-Hui Lin² and Iuan-Yuan Lu¹
¹Institute of Business Management, ²Institute of Information Management
National Sun Yat-sen University, Kaohsiung, Taiwan, R. O. C.

Abstract - A model called PM-Net for representing and monitoring the software development process is presented. This model provides information for project managers, as well as indication of project status at different levels of detail, for the benefit of project managers. This model emphasizes bottom-up data collection and top-down information inquiry functions. The Data Flow Diagram (DFD) and Work Breakdown Structure (WBS) techniques are utilized for construction of a hierarchical structure of the software development process. This process can be viewed as a set of activities, each activity can be viewed as a set of subactivities, and each subactivity can be viewed as a set of tasks. Managers can make an inquiry regarding project progress information at process level, activity level, subactivity level or task level, depending on the degree of detail they require. The manager can also change the control state at any time in the project's running duration, so that managers can monitor the project progress more efficiently.

Index Terms: DesignNet, Petri Net, project progress, project schedule, software project management

Introduction

1. The Main Problems of Software Project Control

Project management includes a mixture of people, resources, systems, and techniques required for carrying out a project to successful completion [6],[13]. The goal of project control is to accomplish the project before the scheduled deadline, within the budget, meeting the specification, and also efficiently utilizing the resources. However, cost overrun, late completion, staff turnover and low quality often occurs in software development projects, especially in large-scale software development operations [6],[14],[15].

Correctly software cost could become a solution to the problem. However, most software projects have the following features which result in difficulties in correctly estimating software cost:

- The unique requirements of each software project;
- The uncertainty of software size estimation;
- The uncertainty of user requirements;
- Relying on software cost estimation is not enough since it is far from accurate and the estimated costs may possibly influence the final project cost [1],[2],[1]. A method is therefore required here which assists the managers in monitoring the actual project progress and in managing the development process.

The lack of an appropriate model for the development of large-scale software has recently been the focus of a great deal of discussion and concern [4],[9],[11]. Lack of control of software project schedule and cost seems to be an international concern. The development of a model for representing the project progress and resource utilization therefore becomes important.

2. The Inadequacies of the Traditional Project Management Models

The Gantt Model [19], The Critical Path Method (CPM) [11] and The Program Evaluation and Review Technique (PERT) [20] are three of the traditional project planning and control models. All of these approaches, Gantt, CPM, and PERT, focus on the scheduling of activities. However, this simple formulation does not capture important characteristics of software projects, with the consequence that the schedule often becomes outdated after the project starts running. In terms of software project schedule control, the traditional models have the following deficiencies of:

- None of these models provide the information that would permit the manager to analyze and draw conclusions regarding the progress of activities [14];
- Current models are inadequate for representing the hierarchical relationship of activities and subactivities as an integral system component;
- Activity dependencies do not include the notion of boolean conditions, even though they handle predecessor activities [14];
- Current models are too coarse to represent the resource schedule when a completed activity is being rescheduled;
- Current models are not capable of providing the information to the manager when an activity is activated before all of its prior activities have been completed;
- Current models are also inadequate for representing the criteria that trigger the start of an activity.

Review of Network Model

The formal definitions and functions of Petri nets, UCLA model and DesignNet are first introduced here in this section. The inadequacies of three models in software project management are also discussed here.

1. Petri Net Model

The Petri net is an abstract model utilized for describing and analyzing information and control flow in asynchronous concurrent system [17],[16],[23].

A Petri net N C = (P,T,I,O) is a finite set of places, i.e., 0. T = [T1, T2, ..., Tn] is a finite set of transitions, m = 0. The set of places and the set of transitions are disjoint. P ∩ T = Ø. II is an input function and O is an output function. Directed are from the places to the transitions and from the transitions to the places are represented by the input and output functions. The input function II is a mapping from a transition Ti to a collection of places P(Ii) known as the input places of the transition. The output function O maps a transition Ti to a collection of places O(Ti) known as the output places of the transition. A marking vector µ = µ(T1), µ(T2), ..., µ(Tn) gives the number of tokens in a place for each place at a particular time. The number of tokens in place p is µ(p) = 1,2, ..., n. A Petri net with a marking µ becomes a marked Petri net, M = (P, T, I, O, µ). Examples of a graph representation of a marked Petri net are shown in Fig. 1.

A Petri net graph consists of two types of nodes. A circle "O" represents a place, where one or more tokens (represented by small dots "*") can reside; a bar "I" represents a transition, which can be fired to move tokens from inputs to outputs.

Fig 1: An example of Petri net

A Petri net executes by firing a transition. A transition fires by removing tokens from its input places and creating new tokens which are distributed to its output places. A transition may fire if it is enabled to do so. A transition is enabled to do so if each of its input places has at least as many tokens in it as are from the place to the transition. Two tokens are shown as an example in Fig. 1 to be initially placed in p₁, p₂, transition T₁ will then be fired and token in p₁ will be moved to p₃. After p₃ has an token, then T₃ will be fired, and two tokens in p₃, p₄ will move to p₁. Once p₄ has two tokens, T₄ will be fired and two tokens in p₃ will be moved to p₁,p₄. Firing a transition can be considered as the happening of an event. It may change the state of the system, creating some conditions to cease before and others to begin to hold.

A place is assumed here to be utilized for representing a planned activity and a token inside it means that this activity is currently activated. Some deficiencies toward applying the Petri net to software project management are:
A token in Petri net graph is only a boolean value conditions. The manager cannot execute how soon that an activity will be completed or started. and cannot judge whether the project's progress has been delayed or not.

(2) The existence of a Petri net is nondeterministic[14]. If more than one transition is enabled at any time, it is not predictable which one will fire first. This makes it more complicated to analyze the properties of a system.

(3) If an executing activity has to be suspended for its responsible staff to work, Petri net also shows the unbalanced input place, such that the manager is still to be informed that the activity is to continue on executing.

(4) Petri net restricting a transition will be fired when each of its input places has at least many tokens in it. But sometimes, for avoiding the project schedule delay, the manager may force to fire the transition to activate the activity when each of its input places has at least many tokens in it. [11]

(5) Petri net cannot help the project manager to distinguish an executing activity whether that activity is a new creation or re-executing.

2. UCLA Graph Model

UCLA is the couple biologic graph model of computation [8][18]. In this model systems are represented by a graph with complex directed arc. A UCLA model is a six-tuple C = (V, A, L, Q, S, F). Where V = {v1, v2, ... Vn} is a set of vertices

A = (a1, a2, a3, ... an) is a set of arc.

L = (L1, L2, ... Lm) is the input(L')

and output(L')

logic mapping for each vertex.

Q = (Q1, Q2, ... Qn); P # A -> N is the output(Q')

and output(Q')

degree of each arc-vertex pair

S = A is the start arc, F = A is the final arc.

UCLA is a kind of model like PERT. It can also be transferred to an equivalent model of Petri net. UCLA utilizes combinatoric logic to control the sequence of operations at the nodes. Tokens are required on each input arc to enable an operation; that is, if the input logic of a node is AND(logic). For OR(logic) tokens are only required on some one input arc. AND output logic, tokens are placed on all output arc, while for OR logic, tokens are placed on any one output arc.

An example of an UCLA graph is shown in Fig. 2. The logic of each arc-node pair is marked on the graph as either:* or *-logic. In this example node can fire whenever arc S has a token. When node F fires, it removes token from arc S and puts token on arc H or arc (AND logic). Node node E on the other hand, will place a token on either arc D or arc R (OR logic). Node node C is enabled whenever there is one token on arc C and one token on arc F. Also, node G is enabled whenever arc C or arc H has a token.

Fig. 2 An example of UCLA graph model

For a project management model, UCLA has some deficiencies like as in Petri net and PERT. Its notation of AND(OR) logic could, however, be transferred to a project management model named PM-Net that is to be proposed here later on, such that it could enhance the functions of PM-Net model.

3. DesignNet Model

DesignNet is a hybrid model, which utilizes AND/or structure operators for the description of the work breakdown structure and Petri Net notation used for representing the dependencies among activities, resources, product, etc., proposed by C. L. Liu and E. Horowitz in 1989 [14]. The waterfall model has also been mapped into DesignNet by them. AND/OR graphs are used in artificial intelligence for modeling a task in terms of a series of goals and subgoals [11]. A DesignNet graph G is defined as five-tuple: G = (P, T, S, O, L). Where P is a set of places, T = (P0, P1, P2, ... Pn). Four place types are defined including activity P0, resource P1, product P2, and status report P3. T is a set of transitions; T = (T0, T1, T2, ... Tm). Transition is in that starts an activity and Tp is a transition after an activity is completed. S is a set of structure operators, S = (S0, S1, S2). Sn is an "AND" structure operator; S2 is an "OR" operator. I is a set of input arc: I = (I1, I2, I3, I4). Ix represents the set of input arc for a transition Tx from a place which is of type product or resource, f1 represents the set of input arc for a transition Tp from a place which is of type activity or status. Ix(Tp) represents the set of input arc for connecting a place node to an operator Sn(x). O is a set of output arc: O = (O1, O2, O3, O4,... On). O1, O2 represents the set of output arc for a transition Tp for Sn(x); O3, O4 represents the set of output arc for connecting an operator Sn(x) to a place node. A DesignNet consists of a set of places, a set of structure operators, and a set of transitions. Structural operators connect places of the same type on two adjacent levels where the lower level places are the decomposition of the higher level places. The hierarchy resulting from the connection of structural operators is the WBS of the project. Pre requisite conditions (products and resources required) before an activity can start and the product generated by an activity are linked to activities by transitions.

A DesignNet functions by firing transitions. Unlike the firing of a transition in Petri nets, which causes tokens to be moved from its input places to its output places, the transition firing in a DesignNet is a more elaborate process. Each transition can be considered as an executable procedure for the sake of handling the transition firing operation. The procedure associated with the transition will be executed when a new token is created in the input place of a transition. This procedure would check against all the input places. The input place, if having active tokens, performs the firing operation by setting input tokens to a consumed state and creating new tokens in output places through the rule in the transition.

DesignNet, despite potentially being a powerful model for software project control, has some deficiencies as follows:

(1) If an activity has several products in its output places, DesignNet cannot display how many of its output products have been completed until all output products are completed. A manager can therefore not know the percentage of completion until it is fully completed;

(2) There are cases such as readjustment of the schedule that an activity has to be activated before each of its input places have at least one active token in it. But according to the transition firing rule, DesignNet cannot handle this condition which often occurs during software development projects;

(3) Different levels of managers require having different kinds of information about project progress, but DesignNet only provides one kind of project progress information;

(4) A current activity will sometimes be forced to become suspended because of staff turnover, requirement specification change, design specification change, etc., DesignNet does not take into account this possibility;

(5) In DesignNet, the token for a completed product will be changed from "active" to "consumed" state once one of its output activity places has an active token in it, such that the product will be prohibited from activating other output activity places. A product can, however, become the input place of more than one activity. This feature of DesignNet does not reflect the actual conditions of software development.

The PM-Net Model

The basic PM-Net concept is first introduced in this section and then it is given a detailed explanation. The implications of PM-Net for software project control are then next discussed here. Certain features of the PM-Net model are then finally clarified.

1. The Basic Design Concept of PM-Net

(1) Providing Information that can Meet Management Requirements

Managers at different levels concern about the percentage of completion and the percentage of budget consumption. Division managers want to know what project is to be completed, what phase of work is being performed, and whether the project schedule has been delayed or not. Project managers must have more detailed information regarding the whole project than what is necessary for the division manager. Bottom-level managers want to know the actual progress of a subactivities or a task. To meet the requirements of different levels of management, we adopted the Data Flow Diagram (DFD) analysis technique [15], [24] instead of the waterfall model [19] to DesignNet. The transition firing rule, the token propagating rule, and the token types have been modified here in some places. This modification is to compensate for the limitations of DesignNet.

For a large-scale software development project, the DFD analysis technique can decompose a project into several distinct processes, and then each project can be decomposed until activities that can be decomposed into a set of subactivities, and even a subactivity can be decomposed into a set of tasks. Two principles have to be followed whenever decomposing project activities and their output projects. These two principles are borrowed from analyzing the business process method [7].

--- 364 ---
(a) Each of the activities at the bottom level of the hierarchy we permitted to have only one output product. If more than one product exists in its output, then these products should be composited into one, or the activity will become decomposed into smaller units again. This occurs owing to the bottom-level Manager being required to have continual access to the status of an activity and its output product for which he is responsible, so that he can make an accurate assessment of project's progress. Based on the rule of firing transitions, if an activity has two or more output products, the manager cannot judge which one of these products will be completed first until all of its output products have been completed. For example, if $a_1$ is a bottom-level activity in PM-Net, and if products $P_1, P_2$ are the output of activity $a_1$ such that $(P_1, P_2) \in G(a_1)$, then we either have to let $(P_1) = (\beta_1, \beta_2)$, such that $P_1 \in G(\beta_1)$, or let $(a_1, a_1, a_2)$ such that $P_1 \in G(a_1), P_2 \in G(a_2)$, and $P_2 \notin G(\beta_1), P_2 \in G(a_2)$.

(b) Each output product, whatever its level in the hierarchy may be, is permitted to be an output for only one activity. If a product is produced by two or more activities in the model, the product should be decomposed, or these activities should be combined into one activity. For example, if $a_1, a_2$ are two activities in PM-Net, and product $P_1 \in G(a_1, a_2)$, then either $(P_1) = (\beta_1, \beta_2)$ has to be decomposed here, such that $P_1 \in G(\beta_1), P_2 \in G(a_2)$, and $P_2 \notin G(\beta_1), P_2 \in G(a_2)$, or $a_1, a_2$ have to be combined into an activity $a_1$ such that $(a_1, a_1, a_2)$, or $P_1 \in G(a_1)$.

These above two principles ensure that the representation of software development activities in PM-Net is precise.

(2) An Appropriate Transition Can be Fired and Enabled Whenever an Event Occurs

A transition can be enabled and fired whenever an event occurs at its input places. An activity token can sometimes be generated and become active before all of its input places are active. An executed activity can also sometimes be suspended, awaiting the completion of a product or awaiting specific technical resources. The model cannot be limited so that a transition can be fired only when each of its input place has at least one active token in it, as this will induce the model for a current status delay in the project's progress.

2. PM-Net Structure

A PM-Net is composed of a six-tuple $D = (P, T, I, O, F, \pi)$ where $P$ is a set of places that is a union of five possible types: $P = \{ D_1, \bar{P}_1, P_2 \}$. A place is a place of status operator, $\bar{P}_1 = \{ \bar{a}_1 \bar{a}_2 \}$, is a set of status operator; $\bar{a}_1$ is a set of status operator, and $\bar{P}_2$ is a place of failure report. $\pi$ is the set of transitions that are a union of two possibilities, $\pi = (T_1, T_2, T_3, \ldots)$. $F$ is the set of input and output arcs for transitions respectively, $F = \{ D_1, P_2 \}$. $O = \{ oils, oils, oils \}$ is the input places of activity $a_1$ through transition $T_1$; $G(a_2)$ is the output places of activity $a_2$ through transition $T_2$. $P_2$ is the place of failure report. All have the same meaning as that defined in DesignNet. $F$ is an expression of the input control state operator which evaluates whether a transition $T_2$ will be enabled or not. $F = (\beta_1, \beta_2)$ is a set of operators modified from the UCLA graph [15]. The process $\beta_1$ is an AND logic that asserts active tokens are needed on $P_2$ as active tokens($\beta_2$), then the transition $T_2$ will be enabled active enabled activity $a_1$. In Fig. 3b, if each of $P_2$ and $P_1$ has an active token, the transition $T_2$ will be enabled and will initiate an active token at $a_1$. If each of $P_2$ and $R_2$ has an active token, and $P_2$ does not, the transition $T_2$ cannot be enabled. In Fig. 3c, the transition $T_2$ will not be enabled until at least one active token in it. In Fig. 3d, if $R_2$ and either of $P_2, R_2$ has active tokens, the transition $T_2$ will be enabled and generate an active token in $a_1$. The type of control state operator at any bottom-level place in the model can be manually changed throughout the duration of project execution. A marking $\mu$ in PM-Net are defined as a mark vector of the number of tokens in a place, its definitions and functions in the model are the same as in DesignNet.

Fig. 3 A control state operator graphic representation

input place to enable the transition $T_2$.* "is an OR logic that means active tokens aren't needed on that input place to enable the transition $T_2$ in currently. The operations "*", "*" being placed at different place will represent different meanings. For example, in Fig. 3a, when each of $R_1$ and either $P_1$ or $P_2$ is active, $T_2$ is enabled.

3. Interconnection of Activities

An interconnection of activities can be defined so as to complete the PM-Net structure at a certain level. The interconnection relation $R$ is a binary relation.

(a) The actual interconnections of activities is made through product $P_2$, such that if $(a_1, a_2) \in R$, and if $P_2 \in G(a_1, P_2 \in \{ \alpha \})$, where $\{ \alpha \}$ means the input places of the activity $a_1$, through transition $T_2$, $G(a_2)$ means the output places of the activity $a_2$ through transition $T_2$ then $G(a_1) \cap G(a_2) = \emptyset$, that is the activity $a_1$ and $a_2$ are connected through an output product $P_2$. The interconnection relations can therefore be defined can be verified here as follows:

$$R_{ij} = \{(a_1, P_2, a_2) \mid P_2 \in G(a_1) \cap G(a_2)\}$$

where $i = 1, 2, \ldots$ and $j = 1, 2, \ldots$

If $P_2 \in G(a_1)$, $P_2 \in \{ \alpha \}$ and $P_2 \in \{ \alpha \}$, if $(a_1, a_2, a_3) \in R$, then $P_2 \in G(a_1)$, $P_2 \in G(a_2)$, will be combined into one product $P_2$ such that $(P_2, P_2, P_2, P_2) \in G(a_1, a_2, a_3)$. To ensure the consistency of the different levels of activities throughout the PM-Net graph, the interconnection relations between different levels activities can be verified here as follows:

First, derived here from the above definition is:

$$R_{1j} = \{(a_1, P_2, a_1) \mid P_2 \in G(a_1) \cap \{ \alpha \}\}$$

$$R_{2j} = \{(a_1, a_2, a_3) \mid P_2 \in G(a_1) \cap \{ \alpha \}\}$$

Then, obtained:

$$R_{ij} = \{(a_1, P_2, a_j) \mid (P_1, P_2) \in G(a_1) \cap \{ \alpha \}\},$$

$$R_{ij} = \{(a_1, P_2, a_j) \mid (P_2, P_2) \in G(a_1) \cap \{ \alpha \}\},$$

where $(P_2, P_2) \in G(a_1, a_2, a_3)$ is the upper level activity of $a_1, a_2, a_3$, and $a_j$ is the upper level activity of $a_1, a_2$.

4. Definitions and Implications of Component Notation in the Model

(1) Token State Types

A transition firing in a PM-Net is a nonvolatile process, as in DesignNet. A token can be in different states at different places. A token in an activity place can be in one of five possible states: active, suspended, complete, failure, complete, discarded. In a product place, a token can be in creating, active, suspended, or discarded state. In a resource place, a token can be in an active, empty, or discarded state. A transition $T_2$ at the bottom-level can be enabled if there is one active token in each of its input places whose control state operator is * for AND logic, as well as an active token isn't needed in its input places with control state operator which is + for OR logic. When a transition $T_2$ is enabled, it will fire and generate one active token in its output "activity" place. When an active token in the active place, then the transition $T_2$ will be enabled, and its output end place will generate a "creating" token. When the token state in an activity is in "complete" or "suspended" state, the token state in its output product place will be in "active" or "suspended" state. A token of a different type in a different place has a different meaning. A token life cycle ranges from "active" or "creating" state to "discarded" state. Since the transition firing is nonvolatile, the number of marking $\mu$ in a place $P_1$ means that the same number of tokens have been generated.

(2) Token Type Notation

According to the token type definition in the prior section, a token in a different state in a different place has a different meaning. The tokens in different states should therefore be represented by different notations. The definition of token type notation is provided in Table 2. By these, a manager can quickly understand the current states of a software development project from the notations of PM-Net represented.
Table 2: The notation of token type

<table>
<thead>
<tr>
<th>token type</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>*</td>
</tr>
<tr>
<td>creating</td>
<td>=</td>
</tr>
<tr>
<td>completing</td>
<td>&amp;</td>
</tr>
<tr>
<td>fail to complete</td>
<td>@</td>
</tr>
<tr>
<td>suspended</td>
<td>@</td>
</tr>
<tr>
<td>empty</td>
<td></td>
</tr>
<tr>
<td>discarded</td>
<td>@</td>
</tr>
</tbody>
</table>

(3). The Priority of Firing a Transition

A transition will be fired in Petri net or DesignNet on the basis of whether all of its input places are in active state or not. No consideration is given to transition firing priority. In the case of PM-Net, more information about project scheduling is hopefully provided here to the software project manager, so that they can better control the project's progress.

Let $T$ be a function indicating the token type of places, and let $I(T_i) = (P_{j1}, P_{j2}, R_{j}) j = 1, 2, 3, ...$, where $P_j$ represents the change specification report, $P_j$ and $R_j$ represent the respective input products and resources of activity $j$. Two kinds of first priorities for the firing of transition $T_j$ are defined here as follows:

(b). $T_{(2)}(a) + \text{match} = \text{compute}$ initially, then

1). $3(T(P) = \text{discarded and } S(R) = \text{**}) \rightarrow T_{(2)}(a) = \text{discarded}$.
2). $3(T(P) = \text{active and } S(R) = \text{**}) \rightarrow T_{(2)}(a) = \text{discarded}$.

That means no matter which state the token in each of the others $T_{(2)}(a)$ is in, the transition $T_j$ will be fired to change the token state in $a_j$ from "complete" to "discarded" state. Such that activity $a_j$ will be reactivated again, and a new version of product $P_j$ that $P_j = T_{(2)}(a)$ will be created again.

(c). $T_{(2)}(a) = \text{active}$ initially, then

1). $3(T(P) = \text{active or } T(R) = \text{active and } S(R) = \text{**}) \rightarrow T_{(2)}(a) = \text{discharged}$.
2). $3(T(P) = \text{active or } T(R) = \text{empty and } S(R) = \text{**}) \rightarrow T_{(2)}(a) = \text{discharged}$.

That means no matter which state the token in each of the others $T_{(2)}(a)$ is in, the transition $T_j$ will be fired to change the token state in $a_j$ from "active" to "suspected" state.

5. Implications of PM-Net in Software Project Control

The PM-Net model can analyze a project's properties for the sake of determining if activities and subactivities are connected, and if the project plan is complete, consistent and well-executed. Moreover, PM-Net has modified some of DesignNet's functions and extended its advantages in the following ways:

(1). The necessary condition for enabling and firing a transition in DesignNet is based on whether all of its input places are active. For a software project, some activities may have begun before all of its input places become active. For example, when the input resource of an activity is active, the project personnel may start working on that activity before other input places become active. Therefore, if the activity is restricted so that it cannot be activated until each of its input places has at least one active token in it, the actual status of software development project will not be represented, especially when an activity has to be activated for the purpose of rejudging the scheduling. The "control state operator" is therefore put here into the input ore of transition $T_j$ so as to modify the transition $T_j$ firing rule of DesignNet. A project manager can control the state of control operator for enabling a transition to put an activity on "active" or "suspected" status. The manager can even change the operator from "AND" to "OR" for the purpose of activating an activity, then change the operator back from "OR" to "AND", depending on the judgement of the manager.

(2). PM-Net utilizes the DFD techniques for decomposing the activities into their bottom-level. The model not only builds up the relationship of an activity's products and resources from the bottom-level, it can also display the relationship of activities, products and resources of any higher level. All of the rules which fire a transition, create a token and activate an activity are applied to the bottom-level of PM-Net. Informations regarding higher levels e.g. project current progress status is propagated from the bottom-level. With PM-Net, a manager can inquire into the information related to project progress from the top-level down to the bottom-level necessary so as to obtain the information required. A top-level manager, for example, may only need to review the overall project's progress, but a bottom-level manager needs to know the executing states of a task. PM-Net provides a feasible representative method for meeting various kinds of requirements for different levels of management. This is the greatest advantage of PM-Net;

(3). Changing the requirements after the requirements specifications have been completed is often requested by the user owing to the nature of a software project. How to accelerate what activities will be affected by a specification change becomes an important issue. In DesignNet, one would start by identifying the token associated with that change and then the project personnel would unters it through the design, implementation and testing activities it triggers, and finally locate the resulting modules. In PM-Net, all one needs to do is to determine what input place in the lowest level for the change of requirements will affect and then the model will automatically trace through the activities and products that could be affected by that change. It will change the place token types for the activities or products that are affected into "discarded" or "suspected" states. The traceability of PM-Net is more powerful than that of the DesignNet.

(4). An activity at a higher-level may have one or more output products. DesignNet cannot represent information involving what output products have been completed and what is still to be created or has not yet started until all of its output products are completed. PM-Net will more precisely represent the project status to a higher-level manager. Therefore, for an activity being executed at a higher level, a manager can be told what output has been completed, what is still created, and what has not yet started being created.

6. Points of Clarification

(1). PM-Net model is an open system in which some tokens are provided at certain places by project personnel who must be able to determine whether an activity has succeeded or whether backtracking to previous activities is necessary. The constraints as compared with DesignNet are resultant because this model depends more on project personnel support in that personnel must determine when and where a control state operator type in the lowest level should be changed.

(2). PM-Net functions by firing transition rules at the bottom-level. The higher-level graphs do not follow the transition firing rules. The token state at any place in PM-Net is reflected by its token state at the lowest-level dependent places. Project personnel cannot change any token state in the higher-level graphs of PM-Net. A PM-Net at the higher-level is only a display graph that represents integrated and summarized information regarding software project progress for the sake of more efficiently assisting higher-level managers to understand and control the project.

Examples

Some examples are provided here for illustrating the functions of PM-Net when applied to software project control. The first level activities of a software project are partially illustrated in Fig. 4. No status operator or failure report are presented in Fig. 4 since $a_1$ and $a_2$ are not the bottom-level activities. Fig. 5 illustrates the second level activities which is decomposed from Fig. 4. The activity $a_1$ is decomposed into $a_3$, $a_1$, and $a_2$, and activity $a_2$ is decomposed into $a_3$, $a_2$, and $a_2$. All of $a_1$, $a_2$, and $a_3$ are bottom-level activities, so there is no control state operator, status operator and failure report in each of the bottom-level activities.

![Fig. 4: Part of first-level activities of a software project](image-url)
state in \( a_{11} \) has to be propagated from its subactivities, which means \( a_{11} \) will also have an active token in it when the token in \( a_{11} \) is active. If \( a_{11} \) have been successfully completed, the project staff would then assign an active token to the successor status operator, and the transition \( T_{22} \) will be fired and the token in place \( P_{22} \) is changed from "creating" to "active" state. Meanwhile the token in \( a_{11} \) will be changed from "active" to "complete" state, and the token in place \( R_{21} \) will be changed from "active" to "discarded" state. If \( a_{11} \) fails to be completed, the project staff then have to assign an active token to the fail status operator, and \( R_{21} \) will create an active token. The respective tokens in \( a_{11} \) and \( R_{21} \) will be changed from "active" to "failure to complete" and "discarded" state, and the control state operator in arc \( R_{21} \) will be changed from "\( * \)" to "\( * \)" state. When the token in \( a_{11} \) is changed to "failure to complete" state, the transition \( T_{22} \) will be fired again, and the token in place \( P_{22} \) will be changed from "creating" to "suspended" state.

As a new active token in resource place \( R_{11} \) is created, to the number of marks \( \mu \) in \( R_{11} \) will be added 1, and the transition \( T_{22} \) will be enabled and fired, then the token in \( a_{11} \) will be changed from "suspended" to "creating" state again. If place \( P_{22} \) and \( R_{21} \) are "active", the transitions \( T_{21} \) and \( T_{22} \) will be fired and will create an "active" token in \( a_{22} \). Then \( T_{22} \) also will be fired and will create a "creating" token in \( P_{22} \). When the token in \( a_{11} \) is "complete", in \( a_{22} \) is "active", then the token in the first level of activity \( a_{1} \) will be in "active" state, and its output product \( P_{22}, P_{24} \) will be in "active" and "creating" states respectively.

Fig. 6 is another example which illustrates the variations of a project's activity progress status in PM-Net. At the beginning, activities \( a_{1} \) and \( a_{2} \) at its input places are observed in Fig. 6a to have no tokens in them. Each of places \( P_{2} \) and \( P_{3} \) are observed in Fig. 6b to have one active token, the transition \( T_{22} \) is enabled, which creates an "active" token in \( a_{22} \), then transition \( T_{22} \) is enabled and creates an "creating" token in place \( P_{22} \). In Fig. 6c, because both of \( P_{2} \) and \( P_{3} \) have active tokens in them, so transition \( T_{22} \) is disabled and the active token in \( a_{22} \) is changed to "suspended" state. Then transition \( T_{22} \) is also enabled and changes the token in \( P_{22} \) from "creating" to "suspended" state. In Fig. 6d, both products \( P_{2} \) and \( P_{3} \) have one active token in them, so transition \( T_{22} \) will be enabled and re-activate the token in \( a_{22} \). Once the token in \( a_{22} \) is active, \( T_{22} \) will be fired and token in \( P_{22} \) will be changed from "suspended" to "creating" state. In Fig. 6e, if one assigns an active token in fail status operator \( b_{2} \), then transition \( T_{22} \) will be fired and an active token in failure report \( P_{22} \) will be created. Once \( P_{22} \) is "active" state, the token in \( a_{22} \) will be changed to "failure to complete" state, then the token in \( P_{22} \) will be changed to "suspended" state and the token in \( R_{2} \) and \( b_{2} \) will be changed to "discarded" state, as represented in Fig. 6f. In Fig. 6g, when a new token is

![Fig. 5 Second-level activities that decompose from Fig. 4](image)

![Fig. 6 An example representing the variation of token states in PM-Net](image)
created in \( R_2 \), the transition \( T_2 \) will be fired again and will change the token in \( a_2 \) to "discarded" state first, then create a new active token in \( a_2 \) again. As \( a_2 \) is active, the transition \( T_9 \) will be fired to change the token in \( P_9 \) to "creating" state, as represented in Fig. 6b. Fig. 6a and Fig. 6c represent the variation of tokens type in some places when an activity is completed.

An example that represents the effect of a requirement specification change on a project's progress is provided in Fig. 7. Only activity \( a_2 \) is observed from Fig. 7a to be executed, all of the others are completed. However, once someone requests to change the requirement specifications, the project personnel will assign a requirement change report for the appropriate site in PM-Net. Once the project staff has implemented the requirement change report \( P_2 \), the input place of activity \( a_2 \), the activity \( a_2, a_3, a_4, a_5 \), and product \( P_4, P_5, P_6, P_7 \) will be affected, and the tokens in all of the affected activities will be changed from "complete" or "active" to "discarded" or "suspended" state. The token in all of the affected products will be changed from "active" or "creating" state to "discarded" or "suspended" state, as represented in Fig. 7b. Fig. 7c represents a new "active" token created in resource place \( R_2 \), which will then cause the respective tokens in \( a_2 \) and \( P_4 \) to be changed from "discarded" to "active" and "creating" state.

\[ \text{Fig. 7 An example representing the effect of requirement changes in PM-Net} \]

**Discussion**

Far too little effort has been spent in consideration of how to aid the managers of a software development process. This is surprising, as so much of what makes large-scale software development distinctive is the large managerial component involved [14]. A software project management model for representing and controlling the rate of progress of an software project in execution has, however, been presented here. PM-Net inherits the power of DesignNet. It can describe and monitor the software development process. Tokens are objects with specific properties. Transition firing is a nonvolatile process and creates new tokens instances providing information relevant to scheduling. Whenever transitions are fired, the project execution history is recorded by the instances created. PM-Net also enhances the power of DesignNet and makes it more suitable for representing and controlling the project's progress. PM-Net utilizes DFD analysis technique for the sake of decomposing activities from top-level to bottom-level. Therefore, PM-Net can represent the project's progress and provide different levels of detailed information for the requirements of different kinds of managers. The modifications that relate to the transition firing function, token types classification, and "control state operators" that put onto the transitions \( T \) input arc further eliminate the inadequacies of DesignNet. The manager can more accurately control software development progress on the basis of this modification. PM-Net will help the manager to make the right decision regarding whether to reallocate software development resources or to adjust the development schedule in the face of problems of schedule delay or cost overrun. PM-Net can basically integrate a vast amount of well organized information for software development control, but it must remain dependent on a data base system. The concept of repository in CASE tools that attempts to place all useful information into a common data base could be wrapped onto the development of a complete environment for software project control system design.

Large software development projects can take many forms and exhibit hybrid patterns of behavior. These development projects must be able to handle all of the possibilities that may occur so as to make such a control system complete. PM-Net therefore seems to be an appropriate model that could be utilized to develop a software project management tool. A framework is currently being developed in our research for software project management with PM-Net as the key feature. The framework will hopefully also become an essential starting point in the development of a fully appropriate system for software project control.

**References**