A graph based process model measurement framework using scheduling theory

Gary Guang-li Mou
New Jersey Institute of Technology

ABSTRACT

A GRAPH BASED PROCESS MODEL MEASUREMENT FRAMEWORK USING SCHEDULING THEORY

by
Gary Guang-li Mou

Software development processes, as a means of ensuring software quality and productivity, have been widely accepted within the software development community; software process modeling, on the other hand, continues to be a subject of interest in the research community. Even with organizations that have achieved higher SEI maturity levels, processes are by and large described in documents and reinforced as guidelines or laws governing software development activities. The lack of industry-wide adaptation of software process modeling as part of development activities can be attributed to two major reasons: lack of forecast power in the (software) process modeling and lack of integration mechanism for the described process to seamlessly interact with daily development activities.

This dissertation describes a research through which a framework has been established where processes can be manipulated, measured, and dynamically modified by interacting with project management techniques and activities in an integrated process modeling environment, thus closing the gap between process modeling and software development.

In this research, processes are described using directed graphs, similar to the techniques with CPM. This way, the graphs can be manipulated visually while the properties of the graphs can be used to check their validity. The partial ordering and the

precedence relationship of the tasks in the graphs are similar to the one studied in other researches [Delcambre94] [Mills96]. Measurements of the effectiveness of the processes are added in this research. These measurements provide bases for the judgment when manipulating the graphs to produce or modify a process.

Software development can be considered as activities related to three sets: a set of tasks (τ), a set of resources (ρ), and a set of constraints (γ). The process, P, is then a function of all the sets interacting with each other: $P = \{\tau, \rho, \gamma)$. The interactions of these sets can be described in terms of different machine models using scheduling theory. While trying to produce an optimal solution satisfying a set of prescribed conditions using the analytical method would lead to a practically non-feasible formulation, many heuristic algorithms in scheduling theory combined with manual manipulation of the tasks can help to produce a reasonable good process, the effectiveness of which is reflected through a set of measurement criteria, in particular, the make-span, the float, and the bottlenecks. Through an integrated process modeling environment, these measurements can be obtained in real time, thus providing a feedback loop during the process execution. This feedback loop is essential for risk management and control.

A GRAPH BASED PROCESS MODEL MEASUREMENT FRAMEWORK USING SCHEDULING THEORY

by
Gary Guang-li Mou

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Computer and Information Science

August 1998

APPROVAL PAGE

A GRAPH BASED PROCESS MODEL MEASUREMENT FRAMEWORK USING SCHEDULING THEORY

Gary Guang-li Mou

Dr. Murat M. Tanik, Dissertation Advisor — Date
Associate Professor of Computer and Information Science, NJIT

Dr. Peter A. Ng, Committee Member — Date
Professor of Computer and Information Science, NJIT

Dr. Douglas D. C. Hung, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT

Dr. Franz J. Kurfess, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT

Dr. Ajaz R. Rana, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT

Dr. Ali H. Dogru, Committee Member — Date
Associate Professor of Computer Engineering Department,
Middle East Technical University

BIOGRAPHICAL SKETCH

Author:	Gary Guang-li Mou
Degree:	Doctor of Philosophy
Major:	Computer Science
Date:	August 1998

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science
  New Jersey Institute of Technology, Newark, NJ, August 1998

- Master of Science in Computer Science
  Florida Institute of Technology, Melbourne, FL, December 1985

- Bachelor of Science in Mathematics and Computer Science
  University of Illinois at Chicago, Chicago, IL, August 1984

- Bachelor of Arts in English Language
  Anhui University, Anhui, People's Republic of China, January 1982

Presentations and Publications:

Gary G. Mou, Murat M. Tanik and Franz Kurfess
"From Project Management to Process Modeling", *Proceedings of the Third World Conference on Integrated Design and Process Technology*, July 1998.

Gary G. Mou and Murat M. Tanik
"A Graph Based Process Representation for Process Modeling", *Journal of System Integration,* Vol. 8, No.2 1998.

Gary G. Mou
"On Road to Process Maturity - Current Research on Software Process Modeling", *Proceedings of the First World Conference on Integrated Design and Process Technology*, Dec. 1995.

Gary G. Mou, Murat Tanik, Sang Suh, and Suzanne Delcambre
"Adding Intelligence to Software Process Modeling", *Proceedings of the 7th Annual International Conference on Expert Systems Applications & Artificial Intelligence*, Nov. 1995.

Gary G. Mou and Sang Suh
"A Firsthand Experience with Expert System Shell, Xi Plus", *Proceedings of the 6th Annual International Conference on Expert Systems Applications & Artificial Intelligence*, Nov. 1994.

Gary G. Mou
Speaker, on the subject of Software Reverse Engineering at the North Texas Federation's annual conference on software engineering, 1992.

To my daughter, Adela.

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Murat Tanik, who throughout the years has been my teacher, advisor and mentor. His guidance has helped to keep my research on the right track and his critiques have ensured the quality of my work. Without his encouragement and reassurance this research would have been impossible.

I am grateful to Dr. Raymond Yeh, whose insights, vision and commitment to the excellence of software engineering have become the basic principle and ground stone for my research.

My appreciation also goes to the other members of my supervisory committee, Dr. Peter Ng, Dr. Douglas Hung, Dr. Franz Kurfess, Dr. Ajaz Rana and Dr. Ali Dogru, for their participation of the committee and their willingness in spending time reviewing my dissertation.

The process of research is a process of building on top of each other. In this respect, I would like to thank Dr. Suzanne Delcambre for her excellent work on process modeling framework using tasks systems templates, and Dr. Steven Mills for his extension of this research. Both of their efforts have provided a solid foundation for my continuation in process modeling.

My special thanks to Mr. Mike Tress for helping me go through the administrative process on campus, so that I could concentrate on my research.

And, finally, I would like to thank my parents who have been giving me love throughout my life and have been praying for me in their own way wishing that one day their son would receive the highest academic degree. My daughter, Adela, has always been an inspiration star in her daddy's heart. To her this dissertation is dedicated.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

LIST OF FIGURES
(Continued)

NOMENCLATURE

Parameters associated with a process:

τ	a set of tasks
ρ	a set of resources
γ	a set of constraints
P	a process instance

Parameters associated with tasks and machine models:

P_{ij}	processing time of task j by resource i
C_{ij}	completion time of task j b resource i
r_j	release date of task j
d_j	due date of task j
w_j	weight of task j
1	single resource
Pm	identical resources in parallel
Qm	resources in parallel with different speed
FFs	flexible flow shop
$\alpha \mid \beta \mid \gamma$	a scheduling problem triplet
$\propto$	a reduction relationship
$>*<$	preemption constraint
$<*$	precedence constraint

NOMENCLATURE (Continued)

Symbols for graphical representation of tasks:

S_E	activity early-start time
F_E	activity early-finish time
S_L	activity late-start time
F_L	activity late-finish time
T	activity time
T_S	activity total slack time

Measurements:

T_p	net process time
C_r	resource capacity usage
C_{max}	make-span
w_jC_j	weighted make-span of task j
L_j	lateness of task j
T_j	tardiness of task j
F_j	float time of task j
F_p	float of a particular path
CI	criticality index (resource tightness)

Stochastic models:

X_{ij}	the random processing time of task j by resource i
$1/\lambda_{ij}$	the mean or expected value of the random variable Xij
R_j	the random release date of task j

NOMENCLATURE
(Continued)

D_j	the random due date of task j
w_j	the weight of task j
$\geq_{st}$	Stochastic dominance

GLOSSARY OF TERMS

The following is a list of the abbreviations used in this dissertation.

BDC	Basic Development Cycle
CASE	Computer Aided Software Engineering
CCS	Calculus of Communicating System
CMM	Capability Maturity Model
CODE	Coding
CPM	Critical Path Method
CRC	Classes, Responsibilities, and Collaborators
CSP	Communicating Sequential Processes
DAA	Design Activity Agent
DAG	Directed Acyclic Graph
DFD	Data Flow Diagram
DODAN	Design Objects Descriptive Attribute Notation
DoD	Department of Defense
FSA	Finite State Automation
FTD	Feature Test Design
FTP	Feature Test Plan
FTX	Feature Test Execution
GUI	Graphical User Interface
HLD	High Level Design

GLOSSARY OF TERMS (Continued)

ICD	Interface control Design
I/F	Interface
IPME	Integrated Process Modeling Environment
IPSE	Integrated Project Support Environment
ISO	International Standards Organization
KPA	Key Practice Area
LLD	Low Level Design
LPT	Longest Processing Time
NJIT	New Jersey Institute of Technology
PAC	Process Analysis Component
PDC	Process Description Component
PDCC	Process Data Collection Component
PERT	Project Evaluation and Review Technique
PI	Process Improvement
PIS	Process Interaction Structure
PMI	Project Management Institute
PML	Process Modeling Language
PSE	Process Support Environment
PTD	Process Test Design
PTP	Process Test Plan
PTX	Process Test Execution

GLOSSARY OF TERMS
(Continued)

SDL	Specification and Description Language
SEI	Software Engineering Institute
SMD	State Machine Design
SPT	Shortest Processing Time
SRS	Software Requirement Specification
TCS	Total Customer Satisfaction
WBS	Work Breakdown Structure
WFM	Work Flow Management
WSPT	Weighted Shortest Processing Time

CHAPTER 1

INTRODUCTION

The software research and development community has come a long way trying to find a "silver bullet" for improving software quality and productivity [Tanik88]. Given the state of research in the current software development environments the "silver bullet" is unlikely to exist in all practical sense. However, improving the software process in producing software products has been widely recognized as the best bullet at hand. Software process modeling is a means to formally capture and describe the process, simulate the process, and improve the process through study and manipulation of data gathered during execution of the process. Software process modeling has gained as much attention in the academic world as software development processes in the software industry [Yeh94].

Process modeling is not a new invention. Process modeling in other disciplines, especially in industrial engineering, has achieved significant results [Hatono92]. However, process modeling in the field of software engineering is yet to be matured. Furthermore, software process modeling by and large remains the topic of interest in the academic world, it has not become part of the development activities in the software development industry. This is, in part, due to the fact that up until now software process modeling research has mainly concentrated on process representation formalisms [Kellner93].

Many of the process representation formalisms can accurately and succinctly capture the existing or proposed software processes, allowing these processes to be studied. These formal process descriptions provide a good foundation for process analysis and improvement. However, describing a process is just a first step in process modeling. The purpose of describing a process is to study the process in order to derive a better process. A better process can be identified through comparison of a set of process parameters. Some of these parameters can be measured quantitatively, in which case direct comparisons are sufficient. However, most of these parameters are neither quantitatively measurable nor qualitatively comparable. Very often, these parameters can be studied in terms of functions, where an increase in one parameter would cause a decrease on the other.

The lack of an industry-wide adaptation of process modeling can also be attributed to the lack of an integrated process modeling environment where process modeling activities can be seamlessly combined with software development activities to create a process centered software development environment. While it is easier to see the output of software development activities than to appreciate the benefit of software process modeling activities, an integrated environment would give the development community the benefit of both.

A process modeling environment should provide the following capabilities [Kellner93]:

- capability of analyzing the described process and identifying deficiency in the process through process simulation;
- capability of identifying alternative processes as a base for comparison and selection based on the constraints and other relevant parameters;
- capability of identifying potential risks through process monitoring and data collections and making corrective recommendations.

The key to these capabilities is process measurement, without which process comparison would have no base for judgment. Thus, establishing a process measurement framework within a process modeling environment as described above is the concentration of this research.

1.1 Research Rationale and Significance of the Research

In addition to satisfying the requirements in pursuing a Ph.D. degree at NJIT, this research intends to address a practical problem from a software development division of a major telecommunications company. This division[1] engages in the development of a satellite based personal communications system for commercial use. Software development of different components of the network is a major part of this endeavor.

This software development organization rigorously follows a collection of development processes established through years of experience in the software development of wireless telecommunications systems. Due to its software development maturity, this division has received an SEI CMM Level 5 certificate (see Chapter 2), one of the three organizations in the U.S. who have achieved this recognition as of the time of this writing. (The other two organizations achieving the SEI Level 5 certificate are Boeing Space & Defense and Lockheed Martin Federal Systems.)

Such an achievement, of course, does not come for free. In order to improve software development quality and to reduce software maintenance cost, the company has

[1] Under this company's proprietary information policy, the company name and the real data being used are not to be disclosed. Instead, some assumed data and procedures are being used in this dissertation.

dedicated enormous efforts in solving the fundamental problems faced by the whole software development industry, especially large software development organizations - process improvement is its number one priority.

Being an SEI Level 5 organization, how can it base its software development activities, including its project management activities, on a process driven environment? How can it effectively utilize data collected through the execution of the process and systematically apply these data back to the process? How can it effectively predict the outcome of its process and, especially, the impact to the outcome due to the volatility and dynamics of the operating environment? A search for a solution to these problems has inspired the work of this research.

This research is an example of research "from the real world and back to the real world". The problem to be resolved is an urgent problem from the software development activities that the author engages in on a daily basis; the result has become a proposal to be submitted as part of the organization's Process Improvement initiative in preparation for the 1998 TCS (Total Customer Satisfaction) regional competition.

1.2 Problem Statement and Research Objectives

Software processes, as Kellner put it, "comprise the technical and managerial framework established for applying people, methods, tools, and practices to the development and evolution of software." [Kellner93] A good *managerial framework* has been established in many organizations and this managerial framework is reflected in their processes. However, as mentioned before, while software development processes have gained much popularity in the software industry, process modeling, as a tool for the *technical* *framework*, remains a topic of interest mainly in the academic world. This separation of the technical framework from its application base diminishes the usefulness of process modeling and hinders research progress in process modeling as well.

The lack of an industry-wide adaptation of process modeling can be attributed to two major reasons: lack of forecasting power in the current software process modeling research and lack of an integration mechanism for the described process to seamlessly interact with daily development activities. Therefore, this research intends to address these two problems: process outcome prediction and process model integration.

1.2.1 Process Outcome Prediction

Process models can be described in a variety of ways, from natural language documentation to syntax sensitive description language to symbolic mathematical formulae. A formal process description mechanism allows the described process to be studied formally. However, the objective of process modeling is not to describe the process, rather, it is to study and manipulate the described process in order to obtain a better process or be able to predict the outcome of the process. Research of process modeling in the past tends to concentrate more on the process description side (see Chapter 2) limiting its application in the real world.

Through the described process, we need to obtain answers to a lot of questions. For example, how are the resources being utilized? Are we over-staffed or under-staffed? Which part of the projects can be done in parallel? Whether or not those parallel activities are necessary or make the most economic sense? Is the deadline realistic? Would adding more resources reduce the task duration? How do the delays affect the project? Where is the bottleneck? Where is the slack time? Where is the critical path? How do we achieve a better resource utilization and reduce the development duration at the same time? Without answers to these questions, project planning and executions are ad hoc.

Answers to the above questions can be quantified and can be obtained through a set of process effectiveness measurements. These measurements are the results from execution of the described processes.

1.2.2 Process Model Integration

Although software development process modeling has achieved significant results, the software development activities in the whole industry, even in the organizations with higher SEI maturity levels, are still not integrated in the process models. Many of these organizations have documented processes in place, but these documented processes mostly serve as policies or guided practice of the software development within the organization, with implementation of these processes supervised by managers or project management personnel.

Software project management, on the other hand, are being emphasized throughout the software development industry. Due to its maturity and simple implementation, most development organizations feel comfortable using project management techniques in managing their daily activities. With the help of project management tools, such as Microsoft Project, or AutoPlan and AutoTeam, project managers are able to keep track of software development activities and produce reports.

After a careful review, though, one can realize that project management tools serve as nothing more than an electronic shorthand for paper and pencil method in using project management techniques: Work breakdown structures (WBS) are recorded in a computer, and different views of the project, such as a Gantt chart, are produced. Schedule modification and progress updates are done by hand.

What seems to be missing is a mechanism which combines project management and process modeling in a cohesive environment where project activities can be manipulated and studied in terms of the established process and processes can be carried out in daily project activities. This research intends to close the gap between project management and process modeling by pulling them together using scheduling as an underlying vehicle.

1.3 Research Approach

The ingredient of software development activities is being considered as consisting of three sets: a set of tasks (τ), a set of resources (ρ), such as development personnel, and a set of constraints (γ). This description of the software activities is analogous to the description of a flexible flow shop (FFs) in the industrial engineering, where a set of jobs have to be assigned to a set of machines with certain constraints.

Scheduling for flexible flow shop situation has been studied in previous works [Pinedo95] in terms of various scheduling models. This research is an attempt to describe software development activities using similar deterministic or stochastic models, and thus provides a basis for theoretical analysis and study of the impact of process to projects and vise versa. The impact can be quantitatively translated to and interpreted with a set of measurements, against which the effectiveness of the process is judged.

The close interaction of software development activities and process modeling calls for an integrated environment where the process is described and simulated, the results are compared, and the execution data are collected.

A sample real world software development and maintenance process is put at test with the framework established in this research.

1.4 Dissertation Organization

After the introduction in this chapter, Chapter 2, Research on Software Development Process Modeling, provides a survey of the current research activities and issues on software process modeling.

One of the concentration of this research is to bring the project management and process modeling together. Chapter 3 (Scheduling, Project Management, and Process Modeling) discusses how these two areas are merged and how the scheduling theory and algorithms are used in a process based project management environment.

Detailed discussions on measurement in scheduling of tasks for a project within a process modeling framework are provided in Chapter 4 and 5. Chapter 4, Measurement of Deterministic Models, deals with static resource allocation and measurements, while Chapter 5, Measurement of Stochastic Models, deals with dynamic resource allocation and measurements.

Process modeling activities cannot be carried out in isolation. Processes have to be described, simulated, measured, modified, and process execution data have to be collected. Chapter 6, An Integrated Process Modeling Environment, introduces such a process modeling environment in which the measurement framework set forth by this research can be carried out.

Chapter 7 provides a conclusion and a list of future research opportunities along the line of this research. Finally, the solution to the practical problem encountered in the telecommunications company mentioned earlier is provided as a case study and is attached as Appendix A.

CHAPTER 2

RESEARCH ON SOFTWARE DEVELOPMENT PROCESS MODELING

In recent years, improving process maturity level of an organization has become a dominant approach for improving software quality and productivity. The technical aspect of the software process improvement calls for the software development process modeling, an effort to drive software development in a process driven environment. This chapter provides a survey of the current software process modeling research. After introducing some terms used in software process modeling and a discussion of the software development focus of the 1990s, the mathematical foundation supporting software process modeling is discussed, followed by a few representative process models and process modeling approaches. Finally, a process modeling framework using task systems is looked in detail.

2.1 In Search of a "Silver Bullet"

The need for establishing a software development life cycle was recognized in late 1960's. As the software systems became larger and more complicated, the software development industry started to realize the need for a process to control the software development. The Waterfall model [Royce70], still in use today, is one of the earliest software development process models.

Since then, the software industry has come a long way on the road of research in the hope of finding a "silver bullet" [Brooks87, Tanik88] to improve software quality, to reduce development cycle time and to reduce costs associated with software development and maintenance. Different approaches have been tried, programming languages have gone through several generations of changes, with each generation at a higher level of abstraction, and a variety of CASE tools are now part of the software development environment that software developers are depending on. While achievements in these areas have helped to improve software quality and productivity dramatically in the past, a level has been reached that further software development improvement cannot solely depend upon further achievement in these areas alone.

Today, still in search of the "silver bullet", the need for another look at the software development process came back - with more stringent requirements, more accurate measurements, and more industry wide practice recommendations. The SEI Capability Maturity Models [Humphrey89] and the ISO 9001 recommendations marked the era of software process development rather than the software product development. It is a widely accepted belief that a mature software development process is an ultimate warranty for a sound software product [Dsn93] [Heim91]. Thus, the software life cycle research has reached a theoretical height - software process modeling.

2.2 Terminology

As in any research area, a multitude of terms emerge as soon as people start the activities in this area, some of which describe the same concept and some misused. Inconsistencies in the use of the terms result in difficulties in communications. This section provides some definitions that are widely used in software process modeling research papers. The terms when used in the context of this dissertation follow these definitions.

Software Process: A software process is the "total set of software engineering activities needed to transform user requirements into operative software and to evolve it" [Jacc93]. It is an instance of a software process model. A *process* differs from a *procedure* in the sense that procedure is defined as a particular way of accomplishing a task while a process is an interdependent set of activities directed towards a particular goal [Singh92].

Some researchers [Ost87] consider a process as consisting of two main components: a *software production process* by which software production activities are carried out, and a *software meta-process* (see below) used to improve and evolve the whole software process.

Software Process Model: A software process model is a descriptive representation of the structure of a software process that is general enough to represent a range of particular processes and specific enough to allow reasoning about them [Jacc93]. It is an instance of a software process framework. A software process model can also be considered as consisting of a *production process model* and a *meta-process model.*

Software Process Modeling: Software process modeling is the discipline of describing and manipulating software process models. The activities of process modeling include process description, process simulation and process analysis.

Process Description: Process description is the activity of capturing a target process, whether an existing process or a proposed one, using some formal or informal mechanisms. An informal process description usually uses a context free natural languages; examples of formal process description mechanisms can be a mathematical formula, a context sensitive process modeling language, or a set of directed graphs, and so on.

Process Simulation: Process simulation is an activity of process manipulation. By interacting with the described process through a user interface, the data associated with the process (e.g. task duration, resource assignments, constraints, etc.) can be modified so that a new process is derived. The process modeling engine provides mechanism to guarantee that the properties of the underlying process description formalism are maintained throughout the process simulation.

Process Analysis: Process analysis is part of the process simulation. With process analysis, the described process is validated and theories and algorithms are applied to the process description formalism to obtain results. These results are measured by data. The data can then be sent back to the process modeling engine for further processing.

Process Enactment: Process enactment in many research articles [Kellner93] [Lehman87] refers to the actual carrying out of the described process. In order to avoid confusion with process execution, usage of the term, process enactment, is avoided in this dissertation. Instead, the term *process simulation* is used.

Process Execution: Process execution in this dissertation means the actual carrying out of the described process.

Software Process Framework: A software process framework is a general outline that defines the fundamental elements, relationships, limits and constraints of a process and the paradigms and protocols for constructing a valid process [Frailey93].

Generic Process: A generic process is something that can be used to generate processes consistent with a software process framework [Frailey93]. Its principal use is in tailoring the process framework.

Production Process: A production process is a process for developing a product.

Meta-process: A meta-process is a process for developing or defining a process [Ost87].

Process Modeling Language: A process modeling language (PML) is a commonly used way of describing engineering activities in the software process.

Process Schema: A process schema provides a template description of a group of process elements, e.g., software production activities, products (artifacts), tools, human roles, projects, organizations, etc. The schema may consist of related sub-schema.

Process Support Environment: A process support environment (PSE), also known as process modeling environment, is a human-oriented, integrated system, intended to allow human beings to interact with computerized tools. It may consist of a process description mechanism, such as graphs or a process modeling language (PML), possibly a library of process schema, and various process tools to support definition, instantiation, evolution, and execution of process models.

2.3 Process Centered Software Development

The focus of the software development industry and the research community has shifted from its concentration on product of software process in the past to the process itself in recent years.

During the 1960s, the focus of the software community was on the coding phase of the development. Most software projects were small, and their success depended on the effective, informal cooperation of small groups of clever programmers. During the 1970s, software projects increased in size, requiring more organized team approaches to the development. The advent of higher level programming languages and the maturity of operating systems freed software developers from implementation details of the software applications and allowed them to concentrate on product development. Software development support tools became available to support the life cycle of requirements, design, integration and test phases. The 1980s elevated the software development from localized activities to a "globally distributed" effort. Information access and information sharing assumed greater importance, and cooperation of multi-site development efforts of large software systems became the topic of the industry. Furthermore, project size continued to increase and the importance of adopting formal, company-wide standards and procedures, and professional management approaches, was increasingly recognized.

To meet the new requirements presented by today's software systems, the software industry in the 1990s is looking into new ways of software development. The search is based on the past software improvement efforts, as outlined in the following four areas. Each of the four areas is typically supported by other existing techniques such as expert systems.

- Computer Aided Software Engineering (CASE)
- Integrated Project Support Environment (IPSE)

- Software development management and control
- New software development paradigms

These efforts provide the working basis for the software process modeling and in turn benefit from the progress in the software process modeling.

CASE tools have long been available to the software development community. The past success in CASE tools development has encouraged continued interest in this area. Along with stand-alone software development tools such as compilers, CASE tools are now available and being used in every phase of the software development life cycle, including requirement specification tools, detailed design tools, automatic test and verification tools, project management tools and configuration management tools.

Availability of CASE tools sparked the idea of integrating the tools into a harmonious software development environment tied together by a software development process. The idea is to automatically manufacture software just like an assembly line would do in an automobile manufacturing company. Thus, "software factories" [Tajima90, Matsu90] became the buzzwords of the software community. In fact, the concept of "software factories" is nothing more than an Integrated Project Support Environment (IPSE), where CASE tools and software development processes are married. Although few of these systems are actually being commercially used [Dsn93], development in this area is likely to revolutionize the software development in the future.

While tools are being developed, management aspect in the software development cannot be overlooked. The guidelines for software development management has been formulated in terms of software development process recommendations as in the Software Engineering Institute (SEI) Capabilities Maturity Model (CMM) and the ISO 9001 CMM.

Specific verifications and checkpoints along with Key Practice Areas (KPA) have been provided in order to help a software development organization achieve a higher process maturity level. The checkpoints are further reinforced by formal inspection processes, such as Fagan Inspections [Fagan86, Hooczko94].

Humphrey has identified and discussed five levels of software process capability maturity [Humphrey88]:

1. Initial - This is an ad hoc process level. At this level, there is no or very little process in place. The operations are chaotic at times and the organization typically operates without formalized procedures, cost estimates, and project plans.

2. Repeatable - At this level, the organization has achieved a stable process by initiating rigorous project management of commitments, costs, schedules, and changes. Because this stability stems from prior experience with similar work, an organization at this level faces significant risk when confronted with changes to product type, tools, methods, organizational structures, etc.

3. Defined - The organization at this level has defined the process as a basis for consistent implementation and better understanding. The process definitions at this level mainly focus on qualitative matters.

4. Managed - Compared to level 3, the organization at level 4 has initiated comprehensive process measurements and analysis, beyond those of cost and schedule performance. Achievement of the managed process level requires a framework for process examination and analysis. Organizations need a means of examining their processes for improvement opportunities in areas such as task workflow, communication mechanisms, task responsibilities, and technology insertion. In addition, this level entails comprehensive

process measurement efforts and their use in process improvements. It is through software process modeling that assistance is provided in identifying and defining such measurements.

5. Optimizing - This is the highest maturity level. At this level, the organization has a foundation for continuing improvement and optimization of the process. At this point, management's focus turns from product improvement to process optimization as a means of ensuring productivity and product quality. At this stage, quantitative data is used to fine-tune the process. Achieving this level requires a mechanism for forecasting the impact of potential process changes in quantitative terms, such as time needed for completion, manpower requirements, or quality. In addition, this level requires a mechanism for recording and analyzing quantitative outcomes of previous process executions and modifications.

As can be seen from the above requirements, a process description mechanism is needed starting from SEI Level 3. It becomes crucial when moving into Level 4, where comprehensive process measurements and analysis are needed. As an organization moves into Level 5, process manipulation becomes an integral part of the software development activities and a vehicle for improvement of productivity and quality. Therefore, a good software process model and process modeling environment is essential to move the organization to a higher SEI level.

One approach to define and execute a software development process is through the *work flow management* technique. Work flow management is a formal approach to process design, operation, and evolution. A work flow is simply an ordered sequence of activities and decision points. Each step and decision in a work flow typically has a single owner, which may be a specific individual assigned some responsibility for an instance of a process (for example, the assigned engineer for a defect report, or the assigned person for an inspection moderator) or a functional role within the project, such as the project manager. Execution can continue from one step to the next only when the owner of a process step has satisfied all of the step's exit criteria.

A process defined through work flow management is easier to be incorporated into a process model for further studies, because the work flow has already been captured, documented, and hopefully, followed. However, the work flow management itself is still a management aspect, unless a technical framework has been established to reinforce the process thus defined. In this case, the technical framework becomes a process modeling environment for a specific process on a limited scope. This is a very good start for a comprehensive process modeling environment.

2.4 Software Development Paradigms

Much debate has been going on in the software development community as to what type of process or life cycle is the best for software development. The Waterfall model is one of the earliest proposed software development life cycle and is still in wide use today. The Waterfall model has the assumption, and requirements, that the exit criteria of a previous phase are completely satisfied before the next phase can start. As such it has received lots of criticism, because the process does not have risk factors built in - a non-perfect exit from a previous phase imposes threat to the quality of the software and/or the schedule of the deliveries.

While the Waterfall model might still be a good process for small systems or certain business applications, such as a payroll system or accounting system, especially when the software development organization has experiences and resources of developing similar systems in the past, it fails to meet the need of large software development.

A large software system development is typically an evolving activity. Many of these systems are developed for the first time and over several years. After the system is completely developed, the phase of enhancement starts. Usually, people (the developers or the customers) have a better understanding of the system only after it has been developed. Software development is a process of creation and experiments. It is a process through which the developers learn by doing. While it is reasonable to expect the developers (and the customers) to know the product as a whole as the system is evolved, it is unrealistic to expect them to know every detail before the system is fully developed. For example, it is almost impossible for the customers to know exactly what they want at the start of the development life cycle, or for the developers to come up with a perfect design (documents, code, etc.) before getting somewhat involved in the next phase of the development. Furthermore, development organizations can't afford to miss the market window in today's highly competitive society a system has to be introduced to the market as soon as possible. After introduction of the system to the marketplace, enhancement to the system can be done based on the use of the system, customers' feedback and developers' experiences with the system.

Therefore, a number of alternative software development paradigms have been advocated, among them the rapid prototyping, the interactive and incremental "stepwise refinement" [Yeh90], and the Spiral Model [Boehm86].

These new paradigms are motivated by a desire to manage the uncertainty in developing the system [Luqi92]. There are at least two different kinds of uncertainty in software development. The first type has to do with the uncertainty as to whether or not a given description is truly a specification of the software to be developed. It has been observed in many software projects that the validation of software specifications is typically completed during maintenance [Luqi92]. The second type of uncertainty has to do with the lifetime of a valid specification. There are three types of changes made to the software: the corrective change (which is 20% of the total number of changes to the delivered software), the adaptive change (25%) and perfective changes (55%) [Dun90]. The purpose of an adaptive change is to adapt the system to a changing environment. Adaptive changes are responses to requirements changes which can be planned or unplanned. Unplanned changes are the most expensive kinds of changes.

To promote the concept of rapid prototyping and stepwise refinement, a research team at Southern Methodist University calls for a four stage system design environment which includes requirements specification, hardware software separation, module specification and system integration [Tanik91, Dogru92, Chris92, Demi92]. To test the concept, a requirements specification environment, DAA (Design Activity Agent), is built supported by a requirements specification language, DODAN (Design Object Description Attribute Notation). The purpose is to provide the designer with the ability to exercise the prototype at a higher level of abstraction, allowing the designer to explore the system behavior in order to gain insights into the original requirements specification and to uncover any new, unforeseen requirements specifications.

2.5 Mathematical Foundation

It is important to build a framework in which processes could be understood, compared, modeled, and improved. We need a systematic basis and a theoretical foundation for performing these activities. In fact, process modeling is deeply rooted from mathematics modeling and is strongly supported by theories and development in mathematics.

Activities in processes, by nature, are like events in a concurrent system and can be abstracted by a dependency graph representing the partial ordering. The dynamic behavior and the non deterministic nature of the processes can be studied in terms of concurrency control. Milner's Calculus of Communicating Systems (CCS) [Milner80] and Hoare's Communicating Sequential Processes (CSP) [Hoare85] are two widely used calculi for modeling concurrent systems and for extracting the semantics of concurrent systems [Tanik91].

The Petri Net model [Petri87] is also a ready host for defining the behavior of various kinds of concurrent systems. The theory of the Petri Net model is derived from the abstraction of a system as a set of casual relationships between events where asynchrony between two concurrent events is a nontrivial relationship. A model similar to the Petri Net model and the CSP is constructed for the system of interacting processes, known as the Process Interaction Structured (PIS) model [Ziegler76]. This model is described as a generalization of the activity-scanning discrete event simulation environment.

Representation of open systems by communicating processes can also be modeled in mathematics. The PIS model can be used to describe the behavior of an object as it is brought to interact with other objects in the open system. The PIS model is thus a system description where all observable behavior of an object is projected as an external behavior of that object. Using the PIS model an observer process can be defined to interact with the system to observe the system behavior from the pattern of events occurring at the interface between the objects and their environment.

Even development in quantum physics sheds lights in software process modeling. Bohm in his book *Wholeness and the Implicate Order* [Bohm80] describes a holistic view of the universe in which the observers are constantly aware of a system of explicit "world lines" representing the perception of the motion of "real" world entities, which, by the perception, are manifest within the implicit order that is the harmonious nature of the whole. The world lines represent a trace of events in a process. A theory formulated by axioms and inference rules can be used to compute the pattern of events of a model and to predict the outcome whenever possible. In classical physics the time-dependent dynamic behavior of a system is specified by differential equations. In quantum physics the time-dependent dynamic behavior of the processes is specified by Schrodinger equations [Sakurai85].

2.6 Process Modeling

Regardless what software process we need to model, process modeling activities typically involve the following three stages:

- Process capturing and description
- Process simulation and measurement
- Process study and analysis

First of all, we need to understand the process. In order to do this, we need to have ways to represent the process abstractions and to set up a process model (process capturing and description). We then need to simulate the process and collect data from simulation of the process (process simulation and measurement). In order to collect data, we need to resolve process measurement issues. The objective is to analyze the process (process study and analysis) by studying the data, comparing different alternatives, simulating different events, etc. These three stages are interactive and recurring, in the sense that after the final stage, the process is modified, re-described, and re-captured, thus the first stage starts again.

It is important to note that a process, as part of software development, will exist regardless of our ability to model it. It is through the modeling that we hope to gain insights into the process itself, and therefore make improvements to it.

The process description research attempts to use mathematics formalisms and formal languages to describe process tasks. The process simulation and measurement research attempts to derive a set of measurements from the task representations to record data collected from process simulation and execution, and to provide feedback to the model for process improvement in the final stage.

2.6.1 Process Models

Processes are normally captured and described in a process model. There are three basic types of process models in terms of their functional concentrations [Kellner93]:

- Descriptive models
- Analytical models
- Prescriptive models

Descriptive models are used to record how some process or class of processes actually were performed, or to characterize hypothetical processes that might be performed. Analytical models are constructed for the purpose of analyzing processes for particular properties such as concurrency or robustness, abstracting only the features relevant to determining those properties. Prescriptive (or normative) models are used to guide, support or enforce the performance of a process, providing advice or instructions on the steps needed to develop a software system. Of course, it would be ideal if a process model can encompass all three functions as described above.

2.6.2 Process Model Formalisms

A software process modeling environment must include some process definition simulation mechanism, or "process engine", and be supplied with process definitions written in some appropriate language or formalism. The following formalisms are often used to describe processes:

Data Flow Diagrams: Using Data Flow Diagrams (DFD) to describe a process may be the oldest and the most straightforward way of representing a process. A directed graph is used with the vertices describing the source or destination of data, or the activities performed on the data, and the directed edges depicting information flow. With DFD, modeling at different levels of abstraction takes place by describing an activity with subsequently more detailed data flow diagrams.

DFDs have been used extensively as a representation technique of functional system models [DeMarco79]. DFDs are good candidates for modeling sequential processes. One of the advantages of using DFDs is that it graphically and intuitively represents data and activities on the data. It is highly data oriented. The limitation of DFD approach is its dependency on the sequential nature of the process being modeled. Its view of processing is static, and therefore not well suited to support process control mechanisms such as iteration and decision making.

Finite State Automation: The Finite State Automation approach is a network-based modeling technique using graphs. A process is described by means of vertices serving as states and edges serving as transitions between states.

There are a number of models described using Finite State Automation approach. The work of Humphrey and Kellner [Humphrey89] puts emphasis on the description of entities, so does the Jackson System Design [Jackson83]. The objective is to describe the behavioral activity of the software process by means of state transitions on persistent entities. By allowing for states to be repeatedly visited, the entity process model accounts for the natural evolution of the software product components. The SP-Machine concept [Armenise89] also uses a finite state machine formalism to describe the software process. The model is augmented with the ability to change the configuration of the machine based on assessments and unforeseen events.

An advantage of using a finite state automata representation as a process model is that its theory is well understood [Hopcroft79]. A deterministic finite state automata, however, is limited in the type of processes that can be described. While any non-deterministic finite state automata can be rewritten, the proliferation of nodes is undesirable for automata of reasonable size and complexity.

Programmatic: Programmatic models are based on the paradigm of high level programming languages and techniques. Process programming represents a non-traditional application domain [Ost87]. The inputs and outputs of the process program are deliverables on the software development life cycle, such as design documentation or integration test cases. The algorithm of a process program specifies the type and order of operations performed by machine or human. Techniques for process program development are similar to software program development and may include steps such as requirements analysis, coding and verification.

This approach stresses the similarities between process definitions and programs, and between process definition simulation mechanisms and program interpretation/execution mechanisms. The potential advantages of this approach include the ability to exploit our existing understanding of how to design programming language-like formalisms and simulation mechanisms for them; and the possibility of adapting existing approaches to program specification, design, and implementation to the specification, design and implementation of process definitions written in a process programming languages. While programming languages may not fully satisfy requirements for process descriptions, they form a basis from which process languages can be researched. Software engineers are comfortable with concepts related to high level programming. Critics of programmatic approach argue that programming languages artificially map a deterministic model to a non-deterministic process [Tully88].

Petri Nets: Petri Nets provide another example of a network-based process modeling formalism [Peterson81, Reisig92]. A most basic form of Petri Net, referred to as "channel-agency nets", provide the ability to describe active components as *agencies* and passive entities as *channels*. The notation uses boxes for agencies, circles as channels, and arrows to indicate a relationship. Basic to the construction of any type of Petri Nets is the requirement that two active components or two passive components are not described in sequence. That is, the order of events will always alternate between passive and active component descriptions.

There are a number of process models using Petri Nets. FUNSOFT [Gruhn91] is a process modeling language which allows for the modeling of control and data flow based on extended Petri Nets. WEAVER [Fernstrom93] uses an extension of Petri Net based formalism to describe activities within a hierarchy of activity types.

Petri Nets are well known formalisms that are used to describe the dynamic behavior of real time processes. Proven algorithms can be used to establish properties of a Petri Net representation. Petri Nets, however, can become unmanageable and incomprehensible for large process descriptions.

Rule-based: In a Rule-based model, a set of expert system rules represents the activities of the process to take place. The ordering of rule enactment is based on conditions previously satisfied. Rule-based process models include SPM [Williams88], MARVEL [Kaiser88] and Prism [Madhavji90].

Rule-based models allow for the non-deterministic description of a process. The ordering of activities is not explicitly stated. Preconditions allow for parallelism to be exploited. The disadvantage of rule-based models is that it may not be a simple task to determine a complete picture of the software process.

Task System Representation: In this approach, a process is represented as a task system [Delcambre94]. A task system is defined as a set of tasks and a precedence relation, denoted by the symbol., <* between tasks which define a partial ordering on the tasks. The precedence relation T<* T' indicates that the operation of task T will complete before the operation of the task, T' begins. More formally the set of tasks describing the process is represented as t={T1, ..., Tn}, and task system is defined as the pair, C=(t, <*). Section 2.7 below provides a more detailed description of the task system.

Table 1 provides a summary of the advantages of the process representation formalisms discussed above. There are also a number of other process representations. Grapple [HL88] uses a set of goal operators and a planning mechanism to represent software processes. They are used to demonstrate goal-directed reasoning about software processes. The Articulator [MS90] describes software processes in terms of object classes and relations, such as task decomposition hierarchies. The defined process classes and relations form formal models of software processes, organizations, and resources, which are used to store process knowledge and simulate process execution [MLS92].

Table 1 Comparison of process representation formalisms.

Process Representation Formalism	**Basic Representation Mechanism**	**Advantages**
DFD	Network-based, directed graph	Process can be graphically and intuitively represented.
FSA	Network-based, finite state automation	Well established theories and algorithms can be applied.
Programmatic	High level programming language	Sophisticated programming logic can be applied.
Petri Nets	Network-based, graphs	Dynamic behavior of real time process can be described.
Rule Based	Expert system rules	Sophisticated expert system rules can be applied.
Task System	Network based, combination of DFD and FSA	Process can be graphically and intuitively represented; well established theories and algorithms can be applied.

2.6.3 Process Model Construction Approach

In addition to using different formalisms to describe processes, the process models themselves are often constructed using different approaches:

Role-Based: The basic abstraction in this model is a "role" played by an agent [CaCo93]. Instead of modeling execution steps, this approach models roles. Managers, designers, and testers are examples of common roles. Each role is listed on a CRC (Classes, Responsibilities, and Collaborators) card. Class is the object class of the role; responsibilities define what a role offers to its community; collaborators enumerate stake-holding relationships between roles. The CRC cards are input to a process evaluation framework in which CRC cards are browsed, clustered (classified, grouped) and animated.

Object-oriented: One extension to the role-based approach is the object-based approach. This approach is based on the view that a development process consists of a number of distinct, concurrent activities, corresponding to the many contributing "roles". These roles and the interactions between them are thus objects that must be modeled. A representative process model using object oriented approach is called IPSE 2.5 [Lonch90].

Force-Based: Another interesting extension to the role-based approach is a force-based approach [Caco93]. Each role is modeled as a charged particle, such that all roles repel each other with a certain strength. This repulsion is balanced by attraction proportional to the strength of collaboration between given pairs of roles.

Expert System Support: Expert systems are used in software process modeling in several ways. As discussed above, software processes can be represented using rule-based approach, as demonstrated in SPM [Williams88], MARVEL [Kaiser88] and Prism [Madhavji90].

Other use of expert systems is to treat models as reusable and sharable resources [KoDo88]. These resources are building blocks that can be used to dynamically build a larger and more complex models under the control of an expert system. A collection of software process descriptions and interdependencies among them are put into knowledge-based process library that supports the organization, access and reuse of software processes [MLS92].

Expert systems are also used to represent knowledge needed to perform a software process, to offer an active assistance to that process [AkMe92]. Thus it is possible to generate process models according to project needs and dynamic modification of the models during the software process.

Hybrid approach: A model developed at Texas Instruments, Inc., named "Tornado Model" [Frailey93] intends to bridge the gap between the Waterfall model phases with the Spiral Model's spiral development activities. The model was inspired by the Spiral Model, but was extended to incorporate more specific milestones and to support concurrent engineering.

The Tornado model encapsulates a Spiral model within a traditional development life cycle. It consists of three levels, the Life cycle, the Phase, and the Basic Development Cycle (BDC). On the top level is the Life cycle, which is the software development life cycle used currently by the company, such as the Waterfall model life cycle or the DOD System Procurement Life cycle, etc. Milestones mark the end of a phase in the life cycle. Within each phase of the life cycle, the Tornado Model calls for a series of basic development cycles, each of which is similar to a single cycle in the Spiral Model. BDCs can recursively generate other BDCs as well. This concept of multiple concurrent parent and child BDCs led to the "Tornado" nomenclature. Within each BDC is a sequence of four stages: Requirements definition and analysis; Design; Implementation; Evaluation and integration.

2.6.4 Process Model Requirements

In order to fulfill the requirements of SEI Levels 4 and 5 (Section 2.3), a process model should have the following capabilities:

- Capability of describing a process formally so that the process can be analyzed mathematically.
- Capability of doing process analysis based on the process description. This includes validation of the process consistency, completeness, and correctness.
- Capability of doing process diagnosis to systematically identify bottlenecks, anomalies, problem areas, and area of opportunities for improvement.
- Capability of doing process comparison based on the input parameters and operational constraints so that a better process can be identified.
- Capability of forecasting the process behavior based on the past performance statistics, current performance data, as well as the process itself, thus risks can be identified and dealt with in a timely manner.
- Capability of adapting to changes quickly so that a new (either modified or improved) process can be put into use as soon as the operational environment has changed.

A process model with these capabilities enables us to produce an improved process with all the constraints of the operational environment. It also enables us to continue monitoring the process for changes.

2.6.5 Software Process Modeling Issues and Outlook

Although research on software process modeling has achieved significant results in recent years, there are still issues that need to be resolved [Dsn93]:

- Most existing process models are based on the Waterfall model life cycle. They lack of capabilities of supporting interactive and incremental development.
- Most existing process models have poor capabilities of adapting to development environment changes. It would be unrealistic to expect that the software development would go exactly as planned. The software process model should have a risk management scheme built in and should be made easier to adapt to the changes.
- Most existing process models only provide recommendations to the software developers. There are no systematic way of enforcing the processes without compromising flexibility.
- Most work today has concentrated on development and experimenting with process modeling notations, and little attention has been paid to the problem of developing systematic methods for capturing processes in those notations.
- Few of the recently developed notation have yet been used to try to define improved processes; rather, they have been used to capture and formalize existing processes or simply to experiment with process definition. There have been attempts to define

better software processes, including processes that emphasize risk reduction. So far, however, these attempts have tended to use informal means of process definition (natural language description or diagrams with informal semantics) and this has made them hard to analyze, improve or follow systematically [Dsn93].

- All software process work is ultimately directed at "software process assessment and improvement", but the term has come to be most closely associated with pragmatic efforts, such as those initiated by the SEI.
- Process enforcement could create inflexibility and data gathering could create operational overhead on the software developers. Inappropriate enforcement of process could slow down software development to the point that the gain would be diminished.

The software industry has come a long way trying to find the "silver bullet" for improving software quality and productivity. While the "silver bullet" is unlikely to exist in all practical sense, improving the software process in producing the software has widely been recognized as the best bullet at hand in the 1990s given the current software development environments. Software process modeling provides means to formally describe and capture the process, simulate the process, and improve the process through study and manipulation of the data gathered during the execution of the process. The process, the model, and the supporting environment form an essential vehicle to carry us forward on our road to process maturity.

2.7 Task System Templates and Resource Models

One of the process modeling frameworks is seen in [Delcambre94]. In this framework, tasks are considered to be un-interpreted units of activity. Resources are accounted for in the model by associating a domain and range with each task. Each task system is described within a generic task system template. The unique aspect of this approach is in its ability to handle the evolution of a process representation as the design of the artifact unfolds by allowing tasks systems to be added or deleted as the artifact structure changes. This allows for the incomplete specification of resources. The process will change as resources are added to and deleted from the project. The product specification will evolve over time as experience is gained and requirements are better understood. The research using tasks systems has been carried forward by Mills in the direction of process reliability and process efficiency [Mills96].

In [Delcambre94], process description is seen at two abstraction levels, corresponding to generic description and dynamic representation. The generic description describes what the process should look like; the dynamic representation of the process evolves as time elapses. The generic description is tailored into a dynamic form allowing specific resources to be allocated to the types defined in the generic description. This is the idea behind the task system templates and the resource models to be described in this section.

2.7.1 Task System Templates

A software process can be represented as a task system. A *task system* is defined as a set of tasks and a precedence relation, denoted by the symbol, <*, between tasks which define a partial ordering on the tasks. More formally the set of tasks describing the process is represented as:

$$\tau = \{T_1, \ldots, T_N\}$$

and the task system is defined as the pair,

$$C = (\tau, <*)$$

where $T_1, \ldots, T_N$ represent the set of tasks.

The task system model concept originally came from the operating systems theory [Coffman73]. Applying the task system model to represent a software process provides basis for analysis of determinacy, deadlock, mutual exclusion, and synchronization between concurrent tasks.

The task system is further expanded into a *task system template* by incorporating other components, such as input, output and measurement. The structure of a task system template [Delcambre94] is shown in Figure 1.

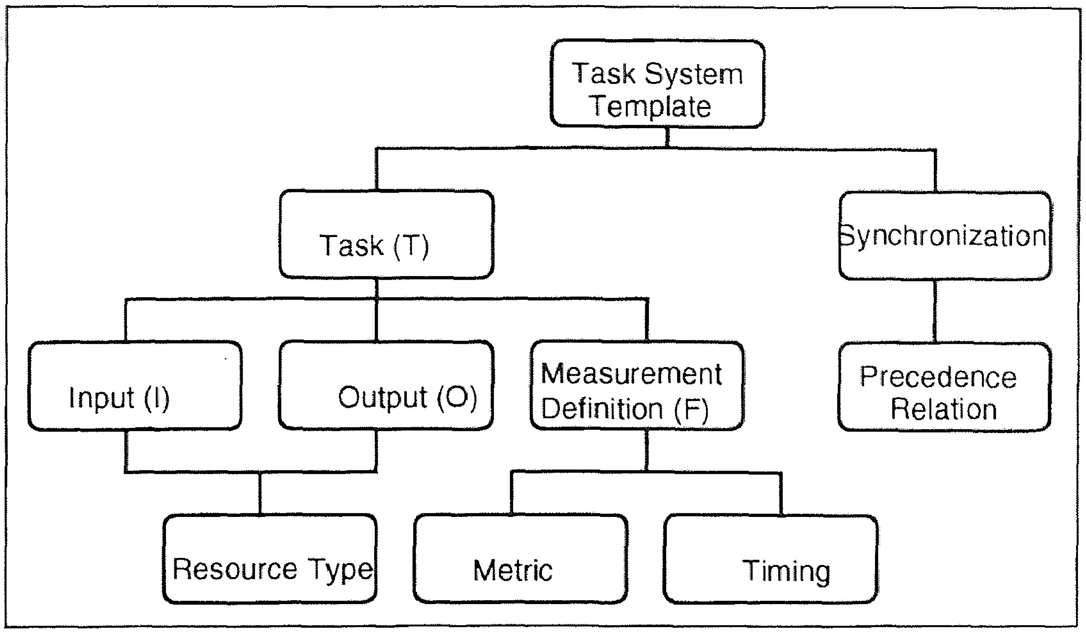


Figure 1 Task system template structure.

A task system template describes a set of tasks and synchronization between sequential tasks. A task, T, is defined by the triple (I, O, F) where I is the set of resource types used as input to the task T, O is the set of output resource types updated by the task, and F is the set of measurement definitions for the task.

In terms of implementation, the precedence relationship of the task systems can be represented with matrix, called *precedence matrix*. This allows convenient study of the task systems by manipulation of the precedence matrix.

The task system model is further enhanced by a generic measurement framework, which allows for metrics to be collected without dictating the specific measurement.

2.7.2 Resource Models

A resource type model can be described as a hierarchy of resource type specifications and includes a set of resource types and the inheritance relationship between resource types. More formally, the set of resource types, ρ, is represented as,

$$\rho = \{RT_1, \ldots RT_N\}$$

and the resource type model, Z, is defined as the pair,

$$Z = (\rho, <\bullet)$$

The relation $<\bullet$ is a partial order showing the inheritance of characteristics from one resource type to another from the set ρ. Thus, $RT_Y <\bullet RT_X$ indicates that the resource type RT_X possesses the characteristics of RT_Y in addition to its own characteristics. Viewing the inheritance relationship between resource types as a partial order allows analysis of the reflexive, symmetric and transitive properties with respect to the relation, $<\bullet$.

A resource type model may be used to represent any resource required to execute the process including personnel, software products and infrastructure elements. One example [Delcambre94] is shown in Figure 2.

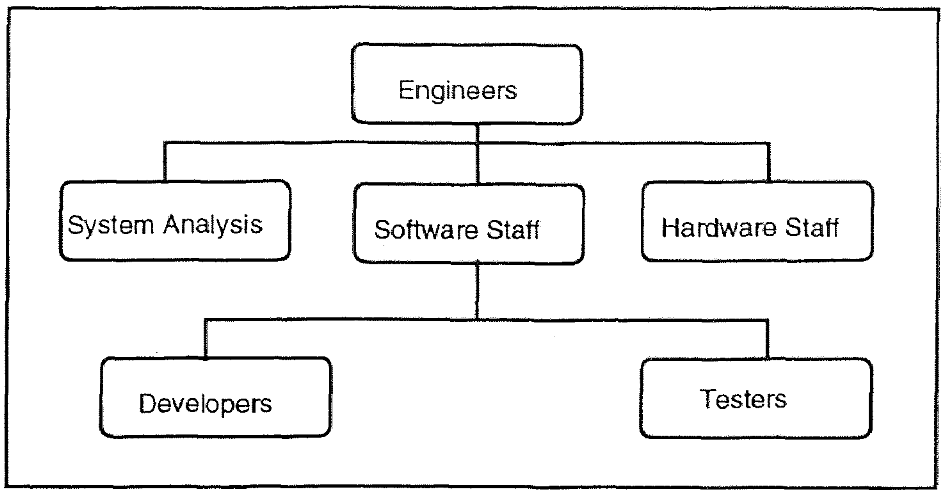


Figure 2 Engineering personnel resource type model.

2.7.3 Process Execution in Task System

In [Delcambre94], the process execution is a process of allocating resources to resource types in the resource model for particular tasks in the task system template.

The execution of tasks in the task system template can be seen as execution of a finite state machine (Figure 3). Each task can be in a particular state, such as *waiting on predecessor*, *waiting on execution, waiting on resources*, etc. depending on whether the required resources are allocated to the task and/or whether the task's predecessor(s) have been executed. The set of task states, σ, is defined as:

σ = {Dormant, Waiting on Resources, Waiting on Predecessor, Waiting on execution, Executing, Suspended, Terminated-Complete, Terminated-Abort}

Fulfillment of a certain task and/or required resources being satisfied trigger a transition of a task from one state to another. Process measurements are taken at particular *collection points*.

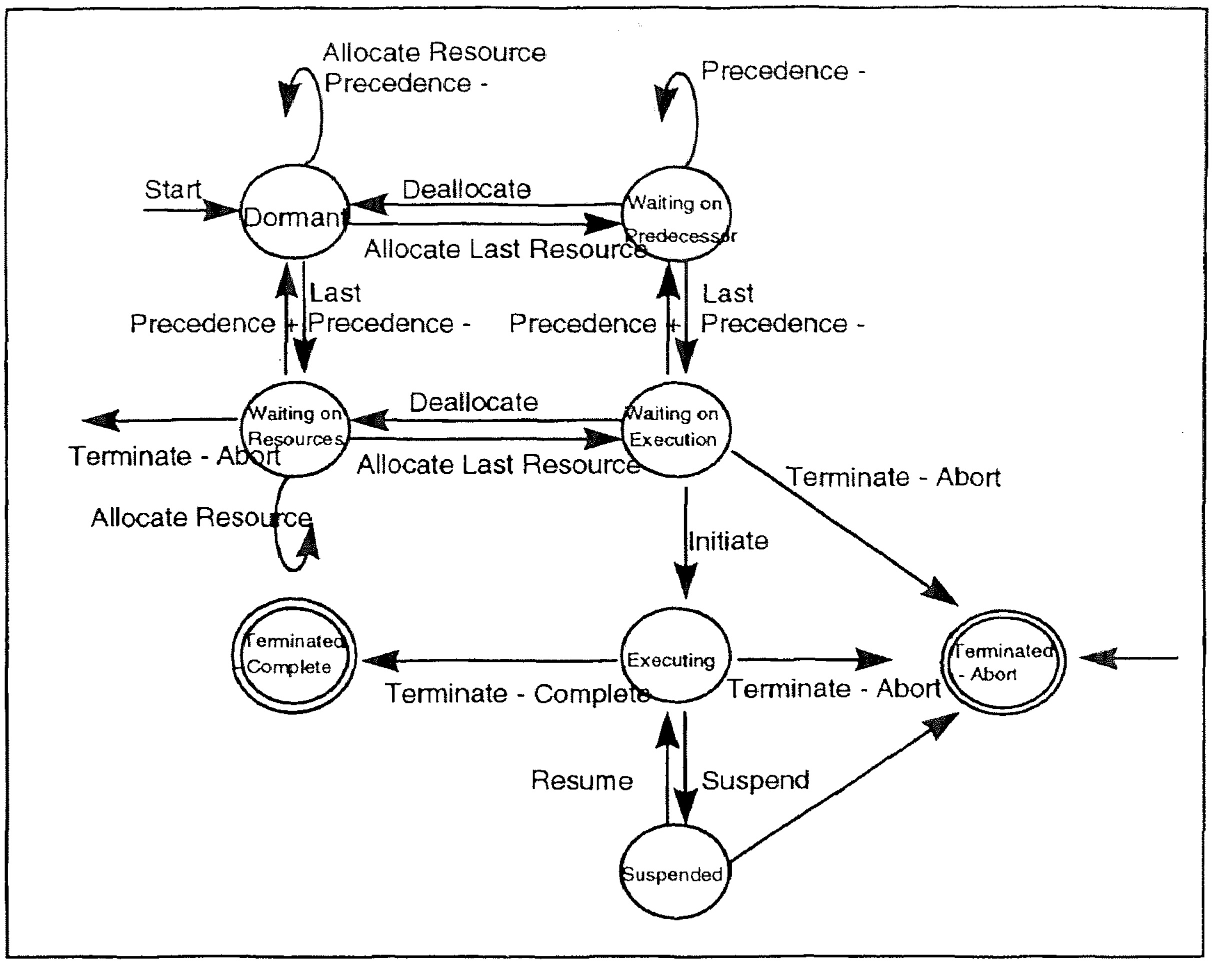


Figure 3 Task state transition mapping.

During task execution, tasks can be further refined by process tailoring operations. As additional activities, alternative processing, or rework of previously executed tasks become necessary during the task execution, either vertical decomposition of tasks or horizontal combination of tasks in the task systems can be performed.

CHAPTER 3

SCHEDULING, PROJECT MANAGEMENT, AND PROCESS MODELING

Scheduling techniques and algorithms were developed half a century ago and are still in wide use today. Example usage of scheduling theories and techniques can be found in manufacturing shop-floor control, in airport gate assignment, and so on. A simple version of scheduling can also be found in an operating system where tasks have to be scheduled.

Books and literature on project management [Moder 83] [PMI96] often cite scheduling techniques in dealing with tasks assignment. Yet these have limited practical significance for two reasons: 1. For a project with a large number of tasks the calculation and maintenance of schedules using elaborated scheduling techniques, such as CPM or PERT, are difficult and time consuming. Therefore, not too many project managers would want to do it by hand. 2. For a project with relative simple set of tasks, the trial and error approach plus some common sense reasoning would be much more effective. Project management software packages, such as Microsoft Project and AutoPlan/AutoTeam, attempt to use scheduling techniques to help the project manager to analyze the tasks and resources assignment, but the application of these techniques is limited to the user interface where the scheduler can visualize the tasks, for example, by looking at the Gantt Chart view of the tasks.

Usage of scheduling techniques in process modeling has been by and large ignored. One reason of this lack of recognition might be due to the fear of labeling process modeling with project management and thus diminishing the work and significance of process modeling. In this research, the call is made and the concept is tested in combining project management with process modeling in a cohesive modeling environment using scheduling theory as an underlying vehicle. This process model based project management environment will push project management to a new and higher abstract level, thus setting the base for process modeling at different organization levels. This chapter discusses a six step approach in integrating process modeling into daily software development activities, including project management and software engineering activities.

3.1 From Project Management to Process Modeling

One of the major trends shaping tomorrow's marketplace is the trend of focusing on process-based management [Tanik 96]. In the software development world, a process-based project management has emerged. Garg has summarized the software development activities into three functions [Garg 96]: process engineering, software engineering and project management.

- Process engineering: Define and maintain software process models.
- Software engineering: Develop and maintain a software product, following a software process.
- Project management: Coordinate and monitor the activities of software engineering.

Process engineering is concerned with the development of process models, which are, in general, independent of particular projects or products. These models are used by project management to create a particular process for a particular project. Software engineering then follows this specific process to produce a product. Figure 4. shows that these three functions are an integrated whole entity.

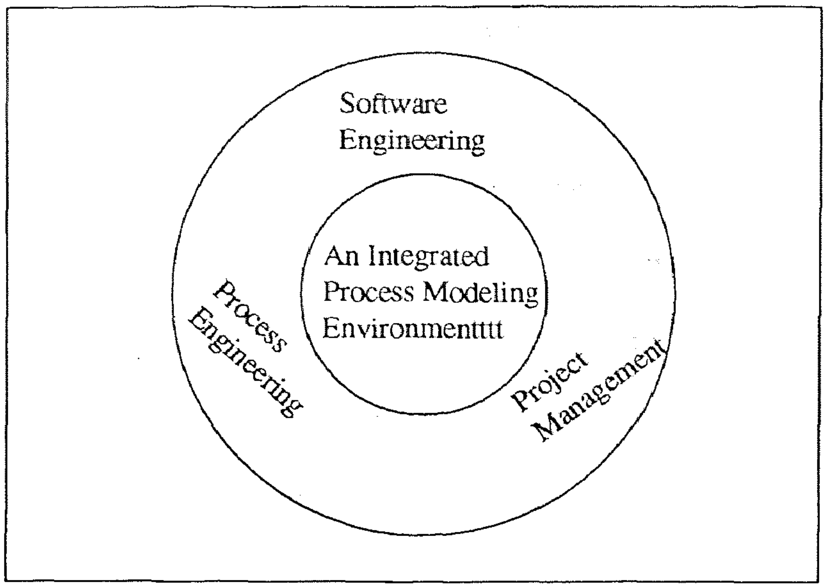


Figure 4 The three functions of software development activities.

Individual tasks in software engineering, which are managed in project management, may have different meaning and play different roles in process engineering, but defining and management of these tasks follow the same principles, which are reflected in the scheduling algorithms. Thus, the underlying principles in scheduling can be readily applied to process modeling.

3.2 Scheduling and Project Management

A *project* is a temporary endeavor undertaken to create a unique product or service [PMI 96]. A project is carried out through various activities and these project activities are managed by *project management* and conducted through *software engineering*.

In order to schedule and coordinate project activities, the activities are typically laid out on a scaled time line, with one bar for each task stretching from the starting time for the task to the ending time [Moder 83]. This is called a *bar chart.* It is based on the technique used by Gantt as early as during the First World War, and it is therefore also referred to as *Gantt chart* (Figure 5).

The primary advantage of the Gantt chart is that the plan, schedule and progress of the project can all be portrayed graphically together. However, it does not explicitly show the dependency relationships among the activities, making it difficult to see the impact on the whole project due to schedule slippage of certain tasks. It does not show resource assignment on the same chart either. Even with the help of a computer, the Gantt chart is essentially a manual-graphical procedure, precluding its usage on a large scaled project with hundreds and thousands of tasks.

Determining a *critical path* in a project is essential in order to accurately estimate the project duration, fine-tune resource assignment and understand the risks involved during the project implementation. This is accomplished with another project management scheduling technique, called *Critical Path Method* (CPM). CPM is a network based

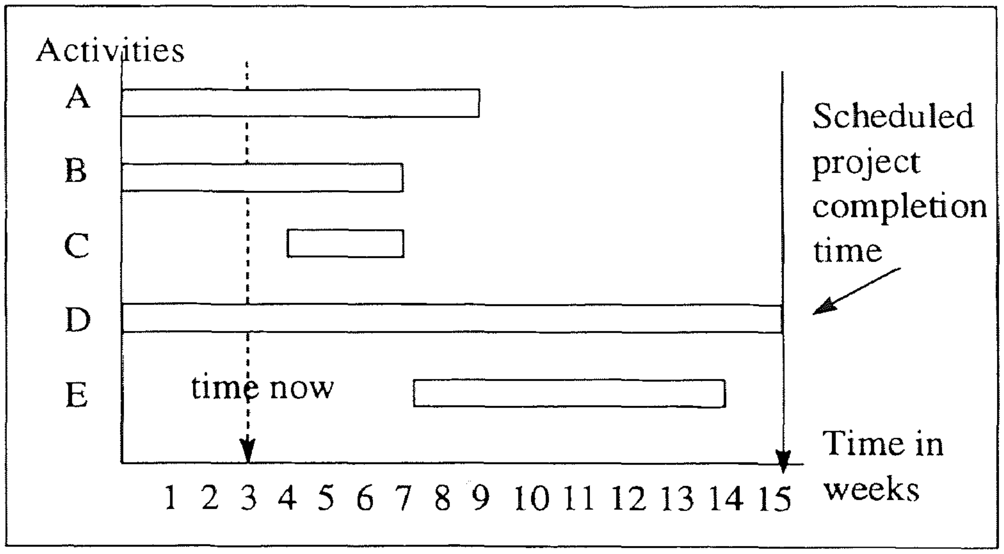


Figure 5 A Gantt Chart

scheduling scheme. The set of tasks for the project are organized in a directed graph, with each node representing an event and each edge representing an activity (or task). The duration of the task is also shown on the graph. Thus, by calculating the longest path from the *source* to the *sink* of the graph, the critical path can be identified (see Figure 6).

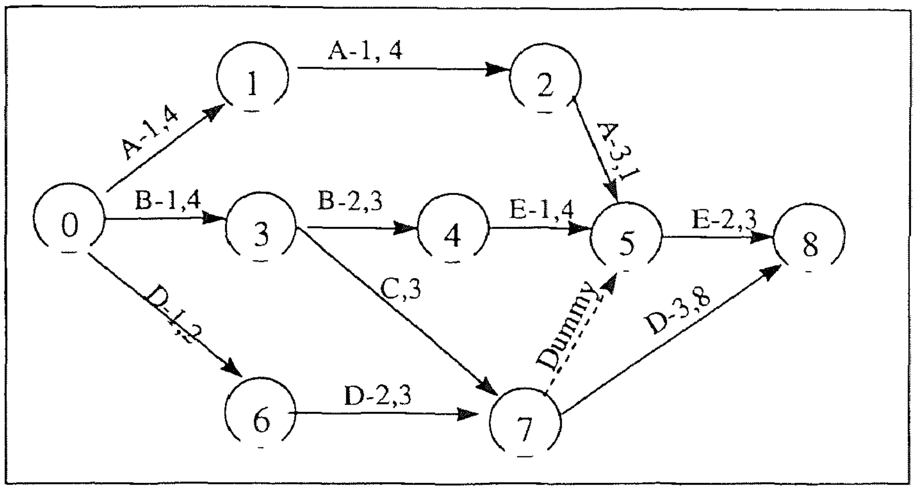


Figure 6 A CPM Network

Critical Path Method facilitates the application of the principle of management by exception by identifying the most critical elements in the plan, focusing management attention on the 10 to 20 percent of the project activities that are most constraining on the schedule. It continually defines new schedules, and illustrates the effects of technical and procedural changes on the overall schedule. Unfortunately, not all the activity duration can be estimated accurately, especially when the past data for this type of activities are not available or the tasks are ill-defined as in many research oriented activities.

PERT (Project Evaluation and Review Technique) overcomes this inefficiency by adding a probabilistic estimate to the activity duration. PERT time estimates are based on pessimistic, most likely, and optimistic time estimates for each activity. Thus the overall project can also be planned using the three different estimates.

Introduced and used by the Navy during the 1950's, PERT is most often used when provisions for measuring uncertainty are important, because of its capability to locate and calculate available slack time using the three time estimates.

Both CPM and PERT provide a systematic and objective approach to planning, scheduling, and controlling projects. They share the following network properties:

- The network diagram can be either "node based" (*activity on node*) or "arrow based" (*activity on arrow*). In activity on node diagrams, each node is an activity (or task); in activity on arrow diagrams, the nodes are events while the activities are marked on the edges (arrows). (See Figure 6.)
- The network has only one start event and one finish event. (For activity on node diagrams, the starting and ending activities may be dummy activities.)
- Both networks are described using *directed acyclic graphs* (DAG) and share the principles of precedence diagrams.
- If an activity on arrow diagram is being used, a network event (node) stands for the completion of all tasks leading to it and no activity may begin at any event until all activities leading into that event have been completed.

3.3 An Information Processing Engine

Process modeling can be described in terms of an information processing engine which produces output based on the input (Figure 7). The input to this processing engine [Kellner 93] is as follows:

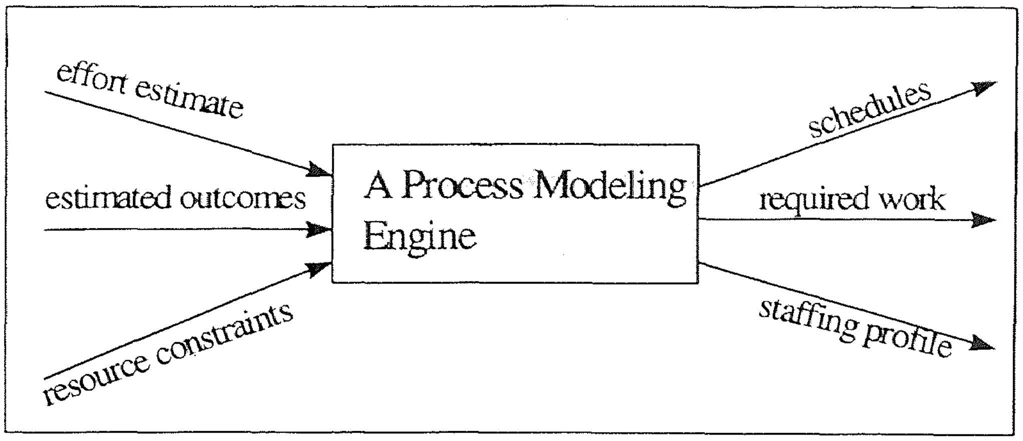


Figure 7 A process modeling engine.

- effort estimate for primitive tasks
- estimated decision point outcomes
- resource constraints

The input to the modeling engine is a set of raw data for the particular project. These data may also include current employee profiles, hardware and software resources, schedules of past projects, etc. These input may have to be normalized, i.e. translated into quantitative entities according to certain rules and formulas so that direct comparison and calculation are possible. With these raw data as input, the engine applies different modeling algorithms and techniques on the data, and produces the following output:

- schedules
- required work effort
- required staffing profiles

The efficiency of the process modeling engine relies on the modeling techniques within the processing engine. To achieve efficient results, the modeling engine should have capabilities of manipulating input data with or without constraints imposed on the data and to produce possible different results based on the constraints. It should also be able to manipulate data that exhibit stochastic properties as well as deterministic characteristics. It should not only be used as a planning tool, most importantly it should also be used as a control tool to control and monitor the process execution, and take corrective action recommendations if necessary.

3.4 Project Tasks and Process Tasks

When integrating project management into the framework of a process modeling environment, it is necessary to distinguish two types of tasks: project tasks and process tasks. One aspect of the project management is to manage execution of the specific project related tasks, referred here as *project tasks*. These tasks are obtained by analyzing the specific project and applying *Work Breakdown Structure* to the project and its sub-tasks. Similarly, work breakdown structure is also used in process modeling. Tasks in the process, referred here as *process tasks*, are also broken down to smaller tasks, and these smaller tasks are further broken down to even smaller tasks as needed. Process tasks are often process execution steps, such as development phases in a software development cycle. These process tasks, along with other data such as constraints, resources, etc., are then used as input to the modeling engine. The modeling engine applies scheduling algorithms and techniques, among other things, to these data and produces results.

3.5 Differences of Project Management and Process Modeling

One may ask, since both project management and process modeling go through the process of work breakdown and both apply scheduling algorithms on the tasks, how does the process modeling differ from project management?

The answer to this question lies in their objectives: The objective of project management using scheduling techniques is to align the project tasks and resources appropriately according to the specified constraints. Execution of these tasks is monitored by the project manager. If needed, a new schedule is worked out and a new execution starts. Even with use of a computer, the practice is essentially a manual application of the scheduling techniques to the tasks obtained from the work breakdown structure. The role of the computer is simply a short hand and convenient way of documenting the tasks arrangement and execution status.

The objective of process modeling, on the other hand, is to map the *project* tasks to the *process* tasks in the context of a process model (Section 3.7.3), and let the modeling engine apply the scheduling techniques to these project tasks automatically, coupled with external intervention by twisting the parameters, such as constraints, resource assignments, and so on during the planning and execution stages, in order to put the project under complete control. The process modeling engine is like a vehicle carrying a specific project. The objective is to carry the passenger (project) to the destination according to the passenger's direction. The role of the vehicle is to monitor the road condition and other environmental constraints, and provide feedback to the driver so that appropriate actions can be taken. Without this process modeling vehicle, the passengers would have to be on their own. They may be given directions on how to get to the destination, as in project management where the schedules and milestones have been laid out, but that's as far as the project management can go.

Kellner points out four differences of project management and quantitative process models as follows [Kellner 93]:

1. Process models are more general and provide enhanced visibility into behavior.
2. Process models highlight importance of feedback loops.
3. Process models are amenable to resource constraints.
4. Process models are amenable to full Monte Carlo simulation.

There are two major characteristics which distinguish a *project* from a *process*, and therefore, project management from process modeling - repetitiveness and abstractness.

Repetitiveness: A project is unique or non-repetitive, while a process is continuous or repetitive. Therefore, project management deals with specific, "one-shot" nature of the project. Project A may be different from Project B. Even though the same techniques and experiences from Project A may be used directly or indirectly on Project B, Project A and Project B are different projects and need to be managed separately. On the other hand, a process modeling deals with repeatable types of activities, regardless of the underlying projects those activities are geared towards.

Abstractness: A project deals with specific activities, while a process deals with *types* of activities. Their work breakdown structures reflect these differences. For example, writing high level document is a type of activity while writhing high level document for Project A is a specific task for Project A.

Let's take a look at two examples: producing a software subsystem for a communications network and producing an automatic speed control software system.

They are two totally separate projects. Breaking down these projects to individual tasks (project tasks) and arranging those tasks in appropriate execution orders require different domain knowledge. Planning and controlling of these projects are the area of interest of project management. However, both projects require a set of execution steps that are common to both, such as designing, coding and testing. These steps, or process tasks, can represent software development phases, such as a high level design phase, or can be broken down further to form more refined tasks, such as writing high level design documents, designing process interfaces, writing interface documents, etc. Planning and controlling of these repetitive and abstract execution steps is the subject of interest for process modeling.

3.6 Interactions of Process Modeling and Project Management

Although project tasks and process tasks are different in terms of their characteristics, management of these tasks shares the same underlying principle and can both benefit from the scheduling techniques. In this sense, the theory and methodology in project management research can be applied, with modifications, to the research in process modeling. It is the intention of this research to bring the two together.

Project tasks can be mapped to process tasks. For example, *coding* is a process task. *Coding of module A* and *coding of module B* are two project tasks that can be mapped to the process task, *coding*. Manipulation of this process task effects the outcome of the project tasks, and scheduling of the project tasks are done though scheduling of the process task. Thus, scheduling techniques and algorithms, which have served in project management, now serve to push the project management to a higher abstract level, and become an integral part of process modeling.

After tasks in process modeling have been defined and refined, a process model has been setup. Tasks in a project can then be mapped to tasks in the process model. Execution of this process model produces execution orders of the project tasks. The collection of data after execution of this process model is fed back to the processing

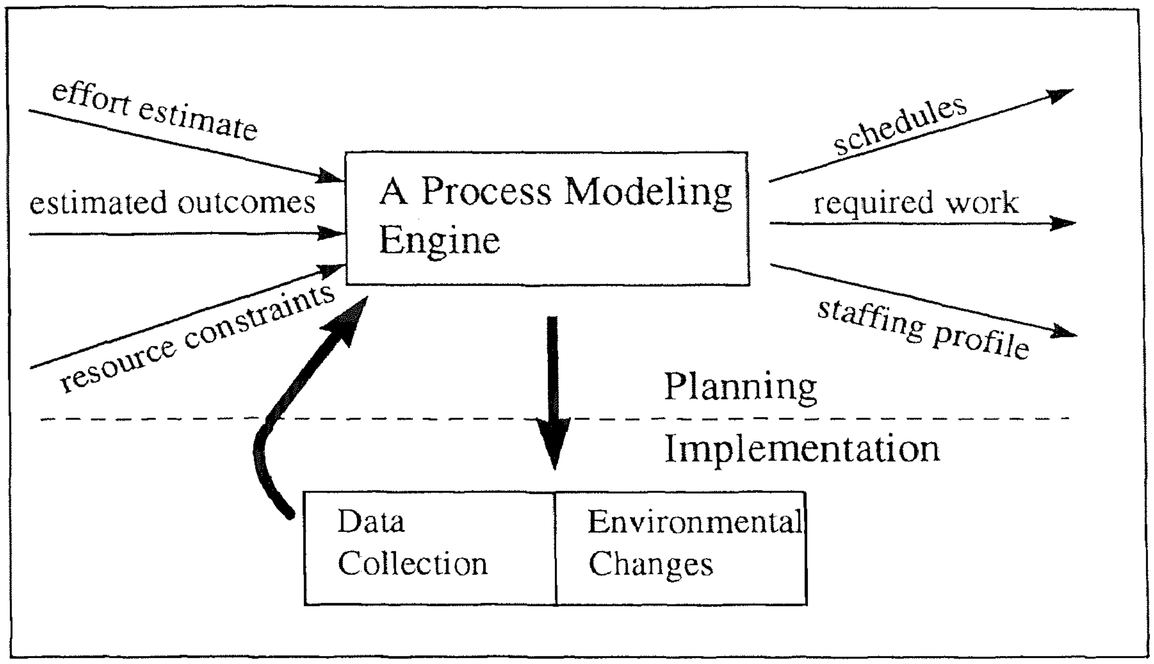


Figure 8 A process modeling engine with feedback loop.

engine. If there are any problems occurring during the project tasks execution, these would be revealed after a re-execution of this process model. Figure 8 shows this feedback loop, where environmental changes and data collected during the process execution phase are sent back to the processing engine for further processing.

This process based project management methodology offers a number of advantages:

1. With process modeling, project management becomes guided and systematic. In order to map project tasks to process tasks, a process has to be in place to guide the activities. The work breakdown structure and the specific activities as documented in project management can then be carried on the process modeling vehicle for their execution. Process modeling thus provides a foundation and framework for project management.

2. Due to the uniqueness and the "one time deal" nature of each project, measurement data obtained in the past cannot be readily applied to project tasks, but can be easily applied to process tasks. Thus, process based project management can be more predictable. Historical data can help accurately estimate the duration of each activity as long as the activity scope, resource assignment, performance history, etc. are known.

3. Due to the interactive nature of the process modeling tool, feedback on project tasks execution can be obtained on a timely basis. New task execution schedule can be obtained interactively. Thus the planning phase and the implementation phase are closely tied together.

3.7 Integration of Project Management and Process Modeling

A process modeling engine provides a framework for project management. It is a vehicle carrying the specific projects to their destination. To better take the advantage of this vehicle, project management and process modeling have to go together.

Figure 9 provides an inside view of a process modeling engine. It includes the following steps:

1. Process model setup.

2. Project work breakdown.

3. Project tasks to process tasks mapping.

4. Project tasks refinement.

5. Process execution and analysis.

6. Process modification based on environmental changes.

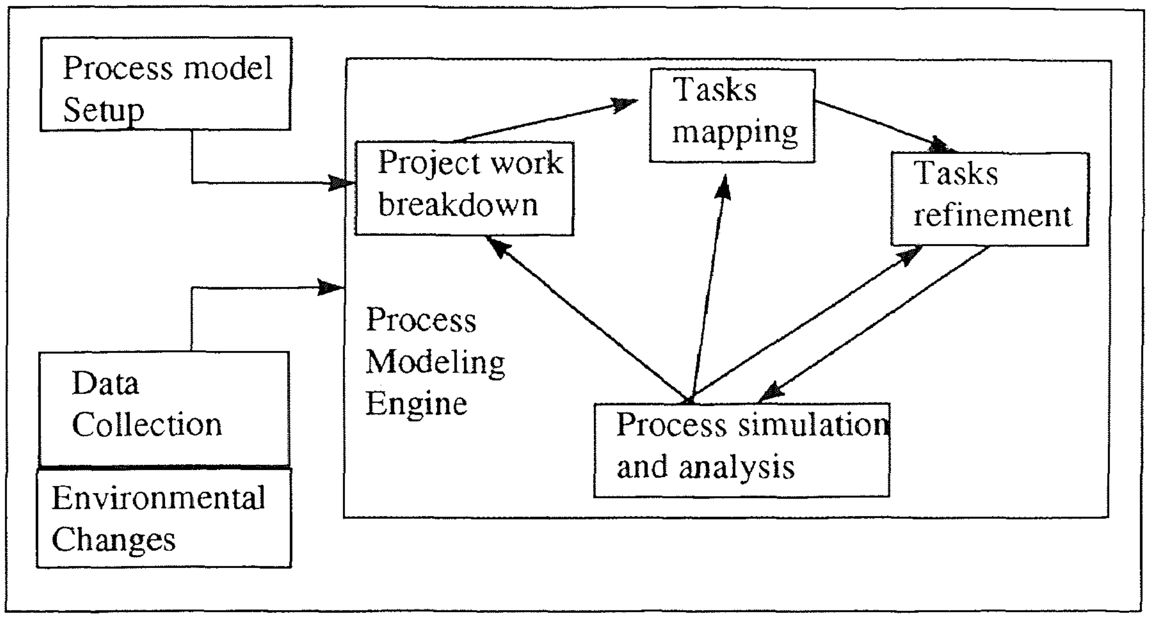


Figure 9 Inside a process modeling engine.

3.7.1 Process Model Setup

Since a process model is the framework for the project management in the process modeling based project management methodology, setting up a process model is the first thing that needs to be done. This process model is a description of the process currently in use or a new process to be followed by the organization. This is a process of obtaining process tasks by decomposing software development phases or product development phases and setting up dependency relationships.

A process model should use formal process representation mechanism, such as a task system [Delcambre 94] [Mills 96], to describe the process tasks in order to conduct theoretic analysis of the process [Tanik 91]. A process model should also allow processes at different abstract levels to be modeled. For example, an organization may want to model the following processes:

A: product concept formulation

B: marketing research

C: product description

D: product specification

E: product development

F: product verification

G: product support and enhancement

These corporate-wide process steps can be further broken down to smaller process steps, or sub-components. For example task E, product development, can be decomposed to component requirements, software requirements, high level design, low level design, coding/unit tests. Coding/unit tests can be further broken down into coding, code inspections, writing process test cases, process test execution, writing feature test cases, feature test execution, writing integration test cases, integration test, etc. Similarly, the product verification task, F, can be broken down to: writing system test plan, system test plan inspection, identifying test tools needed, tools development, equipment ordering, setting up test environment, writing test scripts, test scripts inspections, system tests, test results review, etc.

The level of decomposition depends on which level the process is to be modeled. On a higher organizational level, the decomposition may stop when higher level tasks such as "product design", "product test" are identified. On an implementation level, the tasks may be decomposed into such small tasks that each task is almost like an operational procedure.

When components are decomposed, the dependency relationships have to be updated. This decomposition of components may cause component overlapping or cross-dependency among components.

As seen in the previous list, tasks E and F form a partial order. F cannot be executed until E is finished. If E and F are taken as two sub-components, they can each be further decomposed to form smaller tasks. After the decomposition, there may be tasks common to both E and F, thus the components E and F overlap each other (Figure 10). On the other hand, tasks within E or F may not only depend on tasks in its own component, they may depend on tasks from the other component. This is defined here as *Component Cross-dependency.*

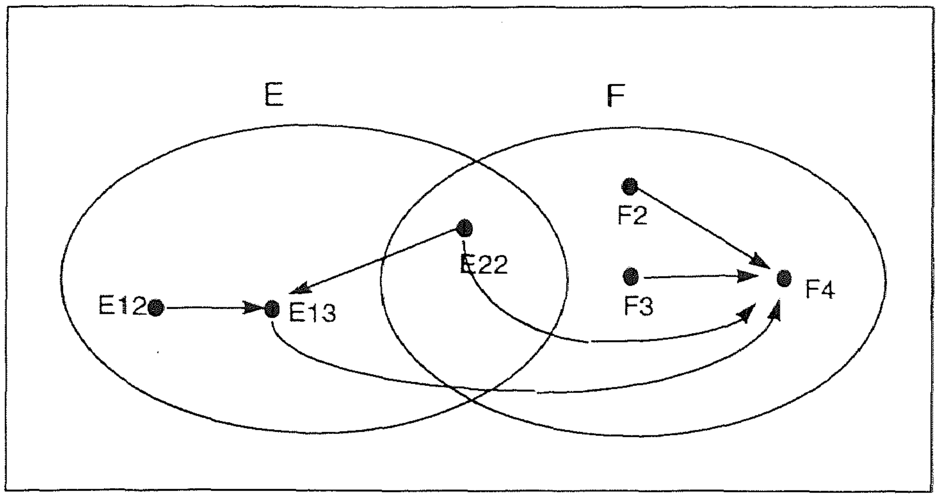


Figure 10 Overlapping components and component cross-dependency.

E12	designer integration test
E13	build load
E22	test equipment setup
F2	system test plan review
F3	prepare system test scripts
F4	system test execution

In Figure 10 the task, test equipment set up, may be common to both component E and component F. In this case, the tasks are merged into one task, E_{22}. Task F_4, system test execution, however, not only depends on task F_3 (prepare system test scripts) it also depends on task E_{13} (build load).

The following list shows the decomposition of Product Development task into sub-tasks. This Product Development task (or phase) consists of the following sub-tasks (Figure 11).

- high level design and documentation (HLD)
- interface design and documentation (I/F)
- state machine design using SDL (SDL)
- process low level design (LLD)
- writing coding (CODE)
- process test plan (PTP)
- process test execution (PTX)
- feature test plan (FTP)
- feature test execution (FTX)

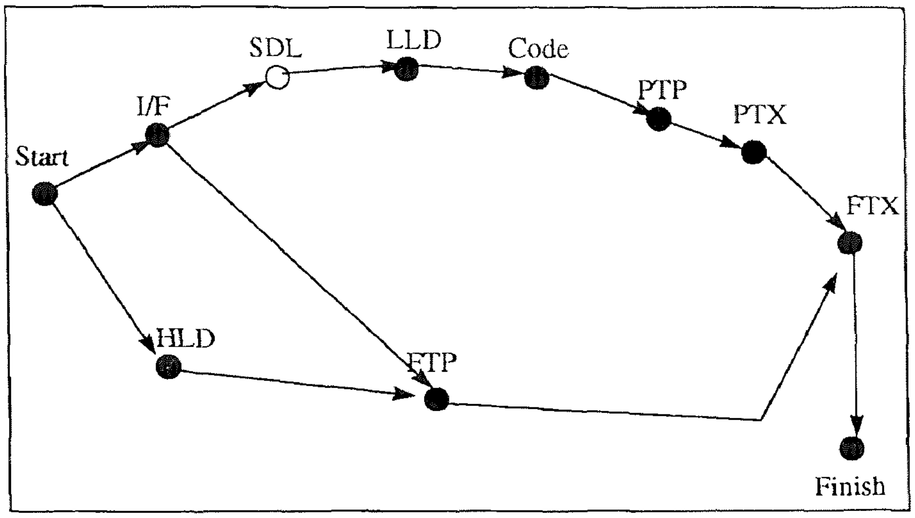


Figure 11 "Product Development" process tasks.

These are *process* execution steps, not *project* execution steps. For example, the LLD step describes low level design activities in the process, not low level design activities for a particular component or a particular module. The low level design for a particular module has to rely on the work breakdown structure as shown in the next section.

Note, in Figure 11 the task SDL is marked with an empty circle instead of a solid one. This means that this step may not be applicable to every situation and therefore is optional.

3.7.2 Project Work Breakdown

Through the application of a work breakdown structure, a specific project can be divided into a number of components. This is a standard practice in software engineering and execution of this step requires domain knowledge of the subject matter and familiarity with the project.

Tasks obtained here are project tasks. Through analysis of the components, a functional dependency relationship, i.e. precedence relationship, for each pair of components is formed. Some components can be developed in parallel with others while other components have to be developed in certain orders. These tasks and precedence relationships, along with their execution duration estimates, are the basic ingredients of a project database.

As an example, a feature in a communications network may involve development work in the following four processes: local connection state machine, transit connection state machine, transcoder resource manager and database manager. Both the local connection state machines require SDL work while the other two processes can start LLD right after the I/F task. Thus, the tasks for this particular feature are defined as follows (see Figure 12):

- HLD for the feature
- I/F for the feature
- SDL for local connection state machine (SDL-L)
- SDL for transit connection state machine (SDL-T)
- LLD for local connection state machine (LLD-L)
- LLD for transit connection state machine (LLD-T)
- LLD for transcoder resource manager (LLD-X)
- Coding for local connection state machine (Code-L)
- Coding for transit connection state machine (Code-T)
- Coding for transcoder resource manager (Code-X)
- PTP for local connection state machine (PTP-L)
- PTP for transit connection state machine (PTP-T)
- PTP for transcoder resource manager (PTP-X)
- PTX for local connection state machine (PTX-L)
- PTX for transit connection state machine (PTX-T)
- PTX for transcoder resource manager (PTX-X)
- FTP for the feature
- FTX for the feature

These tasks are specific *project tasks* for this particular feature. With traditional project management scheme, these tasks would be linked together with one of the project management tools in the form of Gantt Chart, CPM or PERT. Execution of these tasks would be monitored and tracked manually by the project manager.

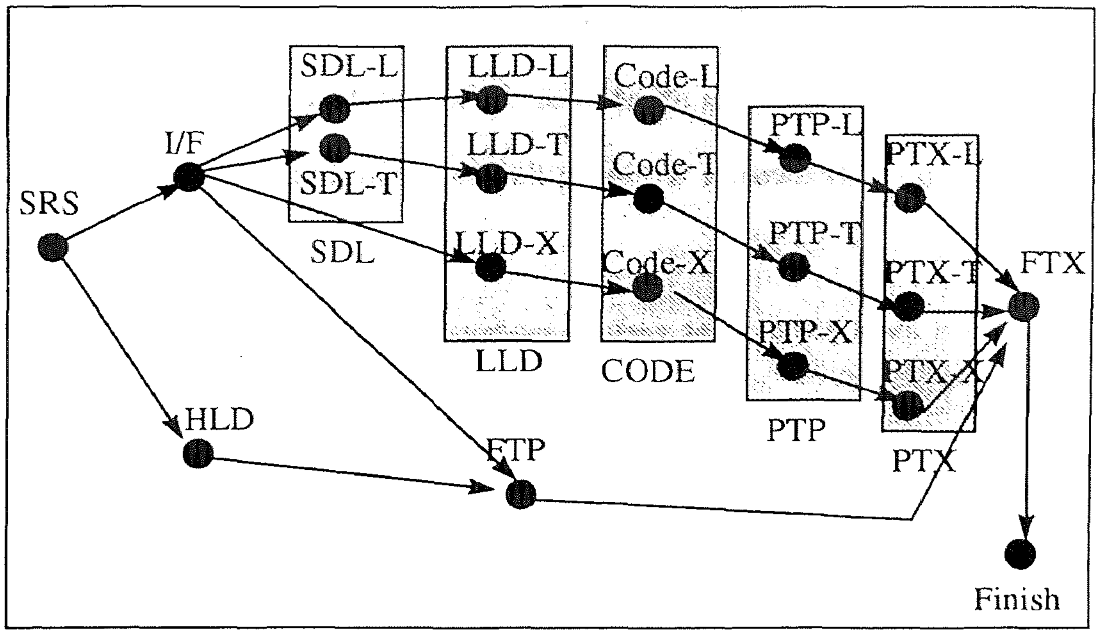


Figure 12 Project tasks mapped to "Product Development" process tasks.

Two important problems need to be considered in breaking down project tasks. One of the problems in defining the project tasks is to determine the scope of each activity. Decisions regarding the scope of the individual activities depend on the level of detail required to plan and control the project, and the resources to be utilized in executing an activity.

Concurrency is another problem. The extent to which the project activities can be performed concurrently, instead of sequentially, will have an important impact on determining the overall length of the project. In one extreme, all the tasks can be executed concurrently if there are no precedence relationships set up, in which case the overall length of the project is the length of time required to complete the longest task. (This "concurrent" execution, of course, assumes that unlimited resources are available to be assigned to the tasks at the same time. Otherwise, the tasks cannot be executed concurrently anyway due to resource dependency.) In the other extreme, all tasks have to be executed in a serial fashion, no concurrency is possible. The overall length of the project is then the sum of the length of all the tasks. In most practices, such extremes are rare, and therefore the length of the overall project is somewhere in between these two extremes.

In a process model based environment, project tasks breakdown is guided by the process tasks breakdown.

3.7.3 Project Tasks to Process Tasks Mapping

Now that both the process tasks and project tasks have been identified in Sections 3.7.1 and 3.7.2, we can map the project tasks to the process tasks, so that project tasks can be executed in the process model.

Figure 12 shows the project tasks being mapped to the process tasks. The original process tasks have been replaced with project tasks, but the precedence relationship of Figure 11 is maintained. In a process modeling environment, these mappings should be done with the help of an interactive graphical user interface, such as GUI.

Two or more project tasks can be mapped to the same process task. For example, SDL-L and SDL-T are both mapped to the same process task, SDL. Both SDL-L and SDL-T need to follow I/F, which is the same precedence relationship as the previous one.

Note that SDL is substituted by SDL-L and SDL-T. SDL-X does not apply and therefore is omitted.

3.7.4 Project Tasks Refinement

After individual project tasks have been identified, duration of execution for each can be estimated, and resources are assigned to each. For a software development process modeling, the resources are either hardware, tools, or people. Assuming that hardware equipment and tools are all (unlimitedly) available, the only type of resources that need to be considered is people. (Availability of hardware and tools is the responsibility of the manager; cost / benefit study of hardware and tools is the interest of project management. Both of which are beyond the scope of study here.) See [Mills 96] for more theoretical treatment of resource assignment.

Constraints are one of the major factors affecting the outcome of the network obtained through this process modeling approach. There are different types of constraints, such as maximum time requirement for the whole project, resource experience level for a particular task. Some of these constraints can be quantified and entered into the system, while others, such as putting people's experience level when assigning resources, can only be used as guidance when manipulating the process model for a project schedule.

Note, at this stage, task refinement is performed without regard to the network outcome which will be discussed in the next section. This initial refinement may lead to a project schedule totally unacceptable at first. Modification of the process occurs at the next stage. The whole modeling process is an on-going and interactive process until a satisfied project schedule has been produced.

3.7.5 Process Execution and Analysis

This is the heart of the process modeling activities. The objective of this modeling stage is to study the graph obtained through the previous stages and calculate such things as overlapping, slack time, critical paths, and so on. The graph is judged against the constraints and measured according to the measurement criteria, *net process time* and *resource capacity usage* [Mou 96]. Further improvement is achieved by manipulating the graph interactively. The process modeling engine ensures that the precedence relationship and the constraints input to the system are not compromised. Further discussions on process measurements are provided in Chapter 4 and 5. An integrated process modeling environment where this process execution can be carried out is discussed in Chapter 6.

There are several types of dependencies: *activity dependency*, *functional dependency*, and *resource dependency*. Activity dependency is inherited from the underlying process model, which is laid out at the beginning (see Section 3.7.1). Functional dependency is laid out at the project work breakdown stage as described in Section 3.7.2. Resource dependency is calculated automatically based on the resource assignment in Section 3.7.4.

Note that activity dependency and resource dependency may conflict with each other. Resolution of these conflicts is one of the primary concerns during this execution and analysis stage.

3.7.6 Process Modification Based on Environmental Changes

Process modeling is an on-going activity. It is because of the dynamic and volatile nature of the project execution environment that makes process modeling based project management a necessity.

During the project planning stage, the project schedule is created interactively in this process modeling environment. A traditional project management scheme would have to stop here as there are no means built-in for systematically monitoring the project execution. However, producing a workable project schedule is just half a story. The biggest difference of the task scheduling in software development compared to that in manufacturing floor control system is that the operating environment in the former keeps changing. People's availability is not written in the stone; estimated tasks duration may be off. All these can lead to either delay or slack time in the schedule. These execution data need to be returned to the process modeling engine for further processing, as shown with the feedback loop in Figure 8.

Recognizing that change is the norm rather than an exception, software project management can no longer rely on the traditional project management techniques alone. Project management tools can record project status and execution steps for a particular project, but they cannot effectively adapt to the rapidly changing environment, in which the software project is carried out. Software management has to go beyond the planning stage automation. A process model based software project management methodology fulfills this requirement.

Scheduling techniques and algorithms, such as CPM and PERT, have played important roles both in project management of software development and in shop-floor control of industrial engineering. They will continue to be the key techniques in process modeling, especially when project tasks and process tasks are integrated in a process drive project management environment.

A process model based project management environment places greater emphasis on the feedback loop during the project execution and promotes the concept of managing by exception. It is more realistic and more adaptive to external changes. It is in this environment that project management is pushed to a higher and repetitive abstract level.

CHAPTER 4

MEASUREMENT OF DETERMINISTIC MODELS

One of the steps in integrating project management and software development activities into a process modeling environment as described in Chapter 3 is process execution and analysis through simulation. This is the time when the parameters (such as resource assignment, duration of tasks, inter-dependencies, constraints) are twisted in order to come up with a better process.

How do we tell that one process is better than the other? In order to do process comparisons, a set of measurements have to be defined. The goal of the process simulation is to experiment with the process parameters and observe the impact to the process as indicated by the measurements.

This chapter discusses measurements resulted from manipulating tasks and resources using scheduling theory and algorithms.

4.1 Scheduling

Scheduling concerns allocation of limited resources to tasks over time. It is a decision making process that has a goal of optimizing or fine-tuning one or more objectives. The resources in a software process modeling environment may be software development personnel, the development machines or test equipment. But here, we only concentrate on resources being personnel, assuming unlimited availability of development machines and test equipment at all times. (Taking machines and test equipment into consideration may be of interest when doing cost analysis, which is beyond the scope of this research.)

The objectives to be achieved may vary. One possible objective could be minimization of the completion time of the last task, and another could be minimization of the number of tasks completed after the committed due dates. Many of these objectives could be in conflict also. Thus, there is no single "best solution" to satisfy all objectives. A *satisfied* process is therefore a process that *closely matches* user's objectives; and a *better* process is a process that has an improvement in achieving one or more of these objectives. The sense of "satisfied" or "better" can be quantified in terms of process measurements.

4.1.1 Static Scheduling and Dynamic Scheduling

Scheduling problems can be roughly classified into *static scheduling problems* and *dynamic scheduling problems*. A static scheduling problem consists of a fixed set of tasks to be performed and a fixed set of resources with which the tasks can be scheduled. A dynamic scheduling problem deals with an ongoing situation with new tasks continually being added to the system. The resources in a dynamic scheduling problem also keeps changing.

To optimize the solution of a static problem, optimization methods can be used. In general, however, optimization methods are only applicable to relatively small problems. The computational difficulty tends to increase exponentially with problem size. Large and complex problems are usually treated with heuristic procedures, such as *dispatching rules* or *sequencing rules*. These are logical rules for choosing which available task to process and by which resource. In using dispatching rules, the scheduling decisions are made sequentially, rather than all at once. This is especially useful when processing times are uncertain, since the rule can decide which task to process next based on those tasks that are actually available to process, rather than those that are supposed to be available. The measurement of efficiency for a static scheduling problem often uses "make-span", i.e., the total time to process all the tasks.

With dynamic scheduling problems new tasks are continually being added to the system. The processing times for these tasks can exhibit either deterministic behavior (see Section 4.1.2) or stochastic behavior. Analytical approaches dealing with stochastic behaviors are often used. These are based on queuing models which provide expected steady state conditions for certain kinds of situations and time distributions. For measurement of efficiency for a dynamic scheduling problem, the emphasis is often on the long-term performance of the scheduling algorithms. The measure is typically on average flow time, the average work-in-process or number of tasks in the system, and resource utilization. The sequencing rules can be used very effectively for dynamic scheduling problems. As the size of the system increases, however, simulation is the most frequently used research methodology.

4.1.2 Deterministic Scheduling Models and Stochastic Scheduling Models

Scheduling is often driven by the nature of the tasks and resources. It is often based on whether the processing time of the tasks and the availability of the resources exhibit deterministic behavior or stochastic behavior. Therefore, scheduling models can be classified into *deterministic scheduling models* and *stochastic scheduling models*. In deterministic scheduling models, it is assumed that the processing time and size of resources are relatively stable. In stochastic scheduling models, the task data, such as processing times, release dates, and due dates, may not be exactly known in advance or they are very changeable. In these models, objectives have to be achieved in expectation. .

This chapter only deals with measurements of deterministic scheduling models. Stochastic models are examined in Chapter 5.

4.1.3 Single Resource Models

A problem of single resource models consists of multiple tasks but only one resource. This is the simplest case where all the tasks are assigned to the single resource. Research on single resource scheduling has been largely based on the static problem of how to best schedule a fixed set of tasks assigned to the single resource, when all tasks are available at the start of the scheduling period.

Since there are relatively few applied examples of single resource scheduling problems, study of single resource models is more useful for gaining insights into the behavior of scheduling rules under particular criteria than for direct scheduling applications.

4.1.4 Parallel Resource Models

A problem of parallel resource models consists of multiple tasks and multiple resources. The development of scheduling procedures for the parallel resources models is much more complex than for single resource models. In the parallel resources models, all resources have to be taken into account to best satisfy the selected criteria.

Both single resource models and parallel resource models apply analytical methods. Due to their complexity the size of problems that can be treated with analytical methods is small and of limited applicability in the "real world". The computer time required to solve scheduling problems with analytical methods grows exponentially with the number of tasks and/or resources to be scheduled. Therefore, a different modeling approach is needed.

4.1.5 Queuing Model and Simulation Approach

The application of queuing models to scheduling problems allows for a relaxation of some of the limiting constraints that are associated with the analytical methods. In particular, the queuing approaches deal with dynamic problems rather than static problems.

To examine realistic, multiple-resource, dynamic scheduling situations, simulation models are most often used. With simulation, one can examine the performance of various rules against several criteria. The size of the problems being studied can be expanded, the effects of startup and ending conditions can be considered, and any kind of task arrival time patterns, or resource capacity, can also be accommodated.

4.2 Scheduling Measurements for Deterministic Models

In all the scheduling problems considered, the number of tasks and the number of resources are assumed to be finite. The number of tasks is denoted by n and the number of resources by m. Let's use the subscript j to denote a task, and subscript i to denote a resource. The pair (i, j) refers to the processing, or operation, of task j by resource i.

In order to understand measurements using different scheduling algorithms, some notations frequently used in scheduling literature are introduced below.

4.2.1 Parameters Associated with Tasks

The following pieces of data are associated with task j.

Processing time (p_{ij}): p_{ij} represents the processing of task j by resource i. The subscript i is omitted if the processing time of task j does not depend on the resource or if task j is only to be processed by one particular resource.

Release date (r_j): The release date r_j of task j may also be referred to as the *ready date*. It is the time the task is ready to be executed, that is, the earliest time at which task j can start its processing.

Due date (d_j): The due date d_j of task j represents the committed completion date (the date the task is promised to be due). The completion of a task after its due date may be allowed, but a penalty could be incurred. When the due date absolutely must be met, it is referred to as a *deadline*.

Weight (w_j): The weight w_j of task j is basically a priority factor, denoting the importance of task j relative to other tasks in the system.

4.2.2 Scheduling Problem Descriptions

A scheduling problem can be described by a triplet.

$$\alpha \mid \beta \mid \gamma$$

α: This field describes the operating environment and contains a single entry.

β: This field provides details of processing characteristics and constraints and may contain no entries, a single entry, or multiple entries.

γ: This field contains the objective to be achieved and usually contains a single entry.

The following sections provide detailed information on what types of parameters can be entered for each one of these fields.

4.2.3 Parameters Associated with Environment and Resources

The following examples are possible environments.

Single resource (1): The case of a single resource is the simplest of all possible operating environments and is a special case of all other more complicated operating environments.

Identical resources in parallel (Pm): There are m identical resources in parallel. Task j requires a single operation and may be performed by any one of the m resources or by any one belonging to a given subset. If task j is not allowed to be performed by just any one, but rather by any one belonging to a given subset, say subset M_j, then the entry M_j appears in the β field.

Resources in parallel with different speeds (Qm): Qm refers to resources in parallel with different speeds; the speed of the resource i is denoted by vi. The time p_{ij}, task j spends with resource i, is equal to p_j/v_i, assuming it is performed only by resource i. If all resources have the same speed, that is, $v_i = 1$ for all i and $p_{ij} = p_j$, then the environment is identical to the previous one.

Flexible flow shop (FFs): This is a term borrowed from industrial engineering used for manufacturing process control. There are s stages in series with a number of resources in parallel at each stage. Each task has to be performed first at stage 1, then at stage 2, and so on. A stage functions as a bank of parallel resources; at each stage task j requires only one resource and, usually, any resource can perform any task. The queues between the various stages usually operate under the FIFO discipline.

Often, an algorithm for one scheduling problem can be applied to other scheduling problems. For example, $1 \mid \mid \sum C_j$ (see Section 4.2.5) is a special case of $1 \mid \mid \sum w_j c_j$ and a procedure for the problem $1 \mid \mid \sum w_j c_j$ can, of course, be used also for $1 \mid \mid \sum C_j$. In complexity terminology it is then said that $1 \mid \mid \sum C_j$ *reduces* to $1 \mid \mid \sum w_j c_j$. This is usually denoted by

$$1 \mid \mid \sum C_j \propto 1 \mid \mid \sum w_j c_j.$$

Based on this concept, a chain of reductions for the operating environments can be established as follows:

$$1 \propto Pm \propto Qm$$

$$1 \propto Pm \propto FFs$$

4.2.4 Parameters Associated with Constraints

The processing restrictions and constraints specified in the β field may include multiple entries. Possible entries in the β field are:

Preemption (><):* Preemption implies that it is not necessary to keep a task on a resource until completion. The scheduler is allowed to interrupt the processing of a task (preempt) at any time and put a different task on the resource. The amount of processing a preempted task already has received is not lost. When a preempted task is put back on the resource (or on another resource, in the case of resources in parallel), it only needs the resource for its remaining processing time. When preemption is allowed, >*< is included in the β field; when >*< is not included, preemption is not allowed.

Precedence constraints (<):* Precedence constraints may appear in single resource or in parallel resource environments, requiring that one or more tasks may have to be completed before another task is allowed to start its processing. There are several special forms of precedence constraints. If each task has at most one predecessor and one successor, the constraints are referred to as *chains*. If each task has at most one successor, the constraints are referred to as *intree*. If each task has at most one predecessor, the constraints are referred to as *outtree*. If no <* appears in the β field, the tasks are not subject to precedence constraints.

4.2.5 Parameters Associated with Objectives

The objective to be minimized can be a function of the completion times of the tasks. The *completion time* of the operation of task j by resource i is denoted by C_{ij}. The time task j exits the system (i.e., its completion time by the last resource) is denoted by C_j.

The objective may also be a function of the *due dates*. The *lateness* (L_j) of task j is defined as:

$$L_j = C_j - d_j,$$

which is positive when task j is completed late and negative when it is completed early.

The *tardiness* (T_j) of task j is defined as:

$$T_j = \max (C_j - d_j, 0) = \max (L_j, 0).$$

Make-span (C_{max}): The make-span, defined as max(C_1, ..., C_n), is equivalent to the completion time of the last task in the system. A minimum make-span usually implies a high utilization of the resources.

In a single resource model, if the objective is to minimize make-span, (i.e. minimize the total time to run the entire set of tasks), it does not make any difference in which order the tasks are executed. In this case the make-span will be equal to the sum of all run times under any sequence of tasks. This is now so in parallel resource models.

Total weighted completion time ($\sum w_jC_j$): The sum of the weighted completion times of n tasks gives an indication of the total holding incurred by the schedule.

The following example shows the parameters discussed above are being used in describing a process model:

FFs | r_j | $\sum w_jC_j$

This notation denotes a flexible flow shop. The tasks have release dates and due dates, and the objective is to minimize the sum of the weighted completion times.

4.2.6 Characteristics of Schedules

In scheduling terminology, a *sequence* usually refers to a permutation of the task set or the order in which tasks are to be performed by a given resource. The sequence does not have to be the same as the precedence relationship (<*), however, the precedence constraints do determine, in part, what the sequence should be. A *schedule* refers to an allocation of tasks within a more complicated setting of operations, which could allow for preemption of tasks by other tasks that are released at later points in time.

A schedule is called *non-delay* if no resource is kept idle when there is an operation available for processing. For most models, including all models allowing preemption, there are optimal schedules that are non-delay. However, it can be shown that there are non-preemptive models where non-delay can cause longer total completion time [Pinedo95]. In this situation it pays to have periods of idleness.

4.2.7 The Measurements for Single Resource Models

The significance of single resource models leads us to the study of more complicated, and more realistic, multiple resource models. The single resource environment is simple and a special case of all other environments. The results that can be obtained for single resource models not only provide insights into the single resource environment, they also provide a basis for heuristics for more complicated resource environments. In practice, scheduling problems in more complicated resource environments are often reduced to sub-problems that deal with single resource. For example, a complicated resource environment with a single bottleneck may give rise to a single resource model.

For the single resource environment, the following theorem has been proved [Pinedo95]:

Theorem 4.2.7: For $1 \mid\mid \sum w_j C_j$ the WSPT rule is optimal,

where WSPT stands for Weighted Shortest Processing Time first. According to this theorem, if tasks are ordered in decreasing order of w_j/p_j, then the total processing time is less.

This is also true for *average time in the system*. If the objective is to minimize the average time that each task spends with the resource, then it can be shown that this will be accomplished by WSPT. As an example, if three tasks with individual processing times of one, five and eight days, respectively, are scheduled, the total time required to execute the entire batch under any sequence is 14 days. If the tasks are processed in ascending order, the average time that each task spends in the system is (1 + 6 + 14) / 3 = 7 days. However, if the tasks are processed in reverse order, the average time in the system is (8 + 13 + 14) / 3 = 11.67 days. This means that the average time in the system will always be minimized by selecting the next task for processing that has the shortest processing time at the current operation.

Average number of tasks in the system: In order to do a better resource planning sometimes it is necessary to know the average number of tasks in the system. The WSPT algorithm also performs well if the objective is to minimize the average number of tasks in the system.

Average task lateness: If the objective is to minimize average task lateness, it can be shown that the Shortest Task First is the best rule for sequencing tasks for the single resource model.

Maximum task lateness: If the objective is to minimize the maximum task lateness, the best sequencing rule is to execute the tasks in due date order, from earliest due date to latest due date.

4.2.8 The Measurements for Parallel Resource Models

A bank of resources in parallel is a setting that is important from both the theoretical and practical points of view. From the theoretical viewpoint, it is a generalization of the single resource model, and a special case of the flexible flow shop. From the practical point of view, it is important because the occurrence of resources in parallel is common in the real-world. Also, techniques for resources in parallel are often used in decomposition procedures for multistage systems.

In this section, the make-span for parallel resource models without preemption is considered. The objective is to minimize the make-span.

With single resource models, the make-span objective is usually only of interest when there are sequence-dependent tasks; otherwise, the make-span is equal to the sum of the processing times and is independent of the sequence. When dealing with resources in parallel, the make-span becomes an objective of significant interest. In practice, we often need to deal with the problem of balancing the load on resources in parallel, and by minimizing the make-span the scheduler ensures a good balance.

We can consider scheduling parallel resources as a two-step process. First, we need to determine which tasks are to be allocated to which resources; second, we need to determine the sequence of the tasks allocated to each resource. With the make-span objective, only the former is important.

Consider the model, $Pm \mid\mid C_{max}$. This problem is of interest because minimizing the make-span has the effect of balancing the load over the various resources, which is important in practice.

However, it can be shown that $P2 \mid\mid C_{max}$ is NP-hard [Pinedo95]. Instead, various heuristic algorithms have been developed for $Pm \mid\mid C_{max}$. One such heuristic is the *Longest Processing Time first* (LPT) rule. It assigns art $t = 0$ the m largest tasks to the m resources. After that, whenever a resource is freed, the largest unscheduled task is put on the resource. This heuristic tries to place the shorter tasks toward the end of the schedule where they can be used for balancing the loads.

At any time, if a set of n jobs (tasks) is to be scheduled on m machines (resources), there are $(n!)^m$ possible ways to schedule the tasks, and the schedule could change with the addition of new tasks. For any problem that involves more than a few resources or a few tasks, the computational complexity of finding the best schedule is beyond the capacity of modern computers. Consider the model with the tasks subject to precedence constraints, that is, $Pm \mid <^* \mid C_{max}$. From the complexity point of view this problem has to be at least as hard as the problem without precedence constraints. To obtain insights into the effects of precedence constraints, a number of special cases may be considered. The special case with a single resource is trivial. It is enough to keep the resource continuously busy and the make-span will be equal to the sum of the processing times. Consider the special case where there are an unlimited number of resources in parallel, or where the number of resources is at least as large as the number of tasks, that is, $m \geq n$. This problem may be denoted by $P\infty \mid <^* \mid C_{max}$. This problem has led to the development of the Critical Path Method (Section 4.4.1) and the PERT (Chapter 5). The optimal schedule and the minimum make-span are determined through a very simple heuristic algorithm: Schedule the tasks one at a time starting at time 0. Whenever a task has been completed, start all tasks for which all predecessors have been completed (i.e., all tasks that can be scheduled).

It turns out that in $P\infty \mid <^* \mid C_{max}$ the start of the processing of some tasks usually can be postponed without increasing the make-span. These tasks are referred to as the *slack tasks*. The tasks that cannot be postponed are referred to as the *critical tasks*. The set of critical tasks is referred to as the *critical path(s)*. To determine the critical tasks, perform the same procedure as discussed above backwards. Start at the make-span, which is now known, and work toward time 0, while adhering to the precedence relationships. By doing so, all tasks are completed at the latest possible completion times and, therefore, started at their latest possible starting times as well. Those tasks whose earliest possible starting times are equal to their latest possible stating times are the critical tasks.

In contrast with $1 \mid <^* \mid C_{max}$ and $P\infty \mid <^* \mid C_{max}$, it can be shown that the problem $Pm \mid <^* \mid C_{max}$ with $2 \leq m < n$ is strongly NP-hard. However, constraining the problem further and assuming that the precedence graph takes the form of a tree (either an intree or an outtree) results in a problem, $Pm \mid Pj = 1, \text{tree} \mid C_{max}$, is solvable. This particular problem leads to the Critical Path rule, which gives the highest priority to the task at the head of the longest string of tasks in the precedence graph.

4.2.9 The Measurements for Flexible Flow Shops

First, let's consider m resources in parallel and n tasks. It can be shown [Pinedo95] that the *shortest processing time first (SPT)* algorithm is still the best choice.

Theorem 4.2.9.1 The SPT rule is optimal for $Pm \mid\mid \sum C_j$.

A somewhat more general modeling environment consists of a number of stages in series with a number of resources in parallel at each stage. A task has to be performed at each stage only by one of the resources. This model is analogous to the manufacturing *flexible flow shop* environment. In this environment, there are s stages in series; at stage l, l = 1, ..., s, there are ml identical resources in parallel. Assume unlimited intermediate storage between any two successive stages. Task j, j = 1, ..., n has to be processed at each stage on any one resource. The processing times of task j at the various stages are p_{1j}, p_{2j}, ..., p_{sj}. Minimizing the make-span and total completion time are referred to as FFs | | C_{max} and FFs | | ΣC_j, respectively. Because this model is rather complex only the special case with proportionate processing times, that is $p_{1j} = p_{2j} = .. = p_{sj} = p_j$, is considered here. With this restriction in place, it can be shown [Pinedo95] that the heuristic SPT rule still produces optimal schedule.

Theorem 4.2.9.2 The SPT rule is optimal for FFs | $p_{ij} = p_j$ | ΣC_j *if each stage has at least as many resources as the preceding stage.*

It can be verified, however, that the SPT rule does not always lead to an optimal schedule for arbitrary proportionate flexible flow shops [Pinedo95].

4.3 An Overview of Scheduling in Industrial Engineering

An important application of the scheduling techniques can be found in the field of industrial engineering as applied in the shop floor control [Hodson 92]. Study of shop floor control sheds light to software development process control.

4.3.1 Shop Floor Control Problem

Shop Floor Control is an information control system maintaining shop orders (manufacturing orders) and work center flows based on the data feedback from the manufacturing shop floor. Its major functions are assigning priority of each shop order; maintaining work-in-process quantity information; conveying shop order status information to the office; providing actual output data for capacity control purposes; and providing measurement of efficiency, utilization, and productivity of the work force and machines. It encompasses the principles, approaches, and techniques needed to schedule, control, measure, and evaluate the effectiveness of shop floor operations.

The efficiency of the shop floor operation is reflected in the following three primary measurements:

- due dates - the assigned or contracted due date for the job;
- flow times - the time that a job spends in the system, from creation or opening of a shop order until it is closed;
- work center utilization - capacity utilization of the expensive equipment and personnel.

Obviously, the goals to optimize the three measurements are conflicting with each other. Seeking a balance or an optimal point to satisfy all three goals becomes the primary concern of the shop floor control system.

Several facets of a scheduling framework in the context of the manufacturing shop floor control need to be considered, for example, shop structure, product structure and work center capacities. In a "*flow shop*" all the jobs tend to go through a fixed sequence of the same routing steps. In a "*job shop*" each particular job tends to have a unique routing, jobs go from one work center to another in a somewhat random pattern, and the time required at a particular work center is also highly variable. Thus the scheduling complexity and constraints in a flow shop can be quite different from those in a job shop. Furthermore, not all the projects can be scheduled the same way. An analysis of the project reveals different ways of organizing the work flow and therefore different schedules. And finally, the work center capacity is another issue. The extent to which the capacities are fixed or variable and the extent to which the capacity for a particular work center can be increased or decreased and the time delay to achieve the change in capacity both affect scheduling performance. If machines and labor are treated as separate resources, the capacity of the work center can either be limited by the machines or by the labor.

4.3.2 Process Control in Industrial Engineering

In comparison to the process modeling in software development the manufacturing process control in industrial engineering has distinguished advantages: In industrial engineering both the capacities of work centers and the time needed for each task are relatively stable and, very often, are well specified or can be determined. While in software process modeling the capacities of the targeted resources are assumed and the task duration of each task is estimated based on some subjective factors. External interruptions, such as absence of the resources can affect the schedule. Therefore, modeling of software processes with human being as major resources are more challenging and the process models tend to exhibit stochastic behavior.

Garg points out [Garg 96]: "What software development and business processes have in common - and what distinguishes them from factory processes which have been precisely defined and automated in assembly-line operations - is that some of the activities in the process are carried out by humans and are intellectual and creative activities. This fact implies that a software process cannot be a static prescription. First, it must be adapted to the needs of the environment in which it is to be applied - for example, based on the expertise level of the team and its experience, or based on resource availability.

Second, it must allow dynamic changes while the process is being executed, in response to changes in the environment, such as the composition of the development team."

4.3.3 Sequencing Rules

A frequent problem for a manufacturing planning and control system is to route the next job to appropriate machine(s) for processing. For example, after processing at machine center A, jobs may be sent for further processing to machine centers B, D, or F. This requires a dispatching rule for choosing the next job in the queue for processing. The question of interest is which sequencing rule will achieve good performance against some scheduling criteria. The following are some well-known rules with their desirable properties:

Random Pick: Pick any job in the queue with equal probability. This rule is often used as a benchmark for other rules.

First Come First Served: This rule is sometimes deemed to be "fair", in that jobs are processed in the order in which they arrived at the work center.

Short Processing Time First (SPT): This rule tends to reduce both work-in-process inventory, the average job completion (flow) time, and average job lateness. There is a concern in using this algorithm. In many studies, this rule has been found to have a higher variance of time in system than other rules. In addition, it can allow some jobs with long processing times to wait in queue for a substantial period of time, thereby causing severe due-date problems for a few jobs.

Earliest Due Date: this rule seems to work well for criteria associated with job lateness.

Critical Ratio: This rule is widely used in practice. The priority index is calculated using (due date - now) / (lead time remaining).

Least Work Remaining: This rule is an extension of SPT in that it considers all of the processing time remaining until the job is completed.

Float Time: A variant of Earliest Due Date algorithm which subtracts the sum of setup and processing times from the time remaining until the due date. The resulting value is called “float” or “slack”. Jobs are run in order of the smallest amount of slack.

Float Time Per Operation: A variant of Float Time algorithm which divides the float time by the number of remaining operations, again sequencing jobs in order of the smallest value first.

Next Queue: A different kind of rule which is based on machine utilization. The idea is to consider the queues at each of the succeeding work centers to which the jobs will go and select the job for processing that is going to the smallest queue (measured either in hours or perhaps in job). The sequencing rule at each work center doesn’t have to be the same.

One important practical result of the research on job shop scheduling has been to clearly understand the combinatorial nature of the problem. The computational costs rise rapidly as a function of problem complexity even when optimal solutions are not being sought. As the number of jobs to be scheduled and the number of work centers increase, the time to prepare a schedule increases much more rapidly. The computational costs of completely simulating all job arrivals and all possible schedules for each machine and worker can be prohibitively high for many actual applications.

4.3.4 Labor Limited Systems

The labor limited scheduling may be of particular interest to software engineering. In many firms excess capacity exists in many machine centers. The controllable cost is labor and the primary scheduling job is how to assign labor to machine centers. A comprehensive framework for the control of work flow in labor-limited systems has been provided by Nelson [Nelson89].

The decision rules suggested by Nelson for determining the availability of a person for transfer utilize a central control parameter, d, that varies between 0 and 1. When $d=1$, the person is always available for reassignment to another machine. When $d=0$, the person cannot be reassigned as long as there are jobs waiting in the queue at the person's current work center assignment. The proportion of scheduling decisions in which a person is available for transfer can be controlled by adjusting the value of d between 0 and 1.

4.4 Application of Scheduling to Process Modeling

This section discusses how scheduling theory can be applied to the software development process modeling.

4.4.1 A Sample CPM Network

Suppose we want to design a small feature in a communications network, which involves three processes, the local state machine, the transit state machine and the configuration management (database) process. The development is to follow the established software development process of the company which has become the basis for this project tasks scheduling in the process modeling engine. Project tasks have been defined and mapped to process tasks as described in Chapter 3. Given the following list of project tasks, Figure 13. shows their execution sequence.

Task	Task Description	Duration (weeks)
A	System Specification	3
B	Interface Control Document	2
C	Software Requirement Specification	1
D	High Level Design	4
E	Implementation on LCSM	3
F	Implementation on TCSM	2
G	Database Setup	1
H	Test on State Machines	4
I	Integration Test	2

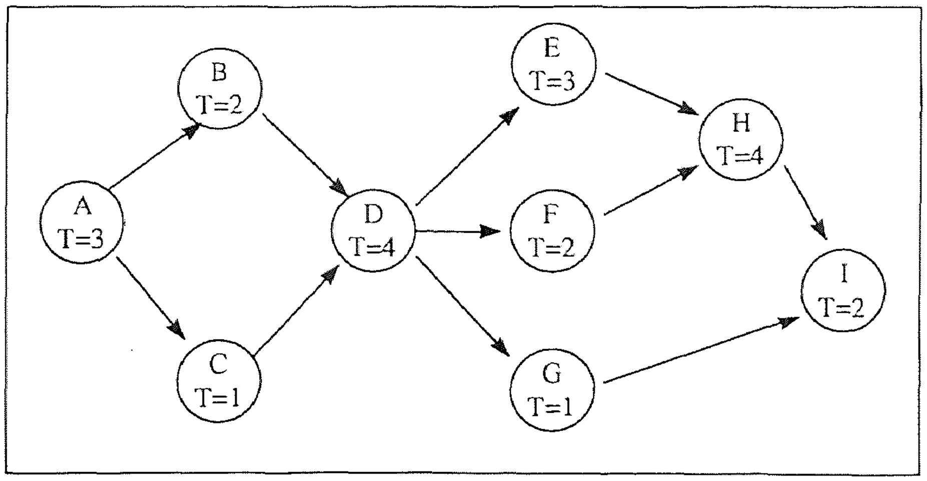


Figure 13 An example tasks execution sequence.

This CPM network diagram for this project begins with the engineering activity, system specification (A), and ends with the task, system integration (I). Once activity A has been completed, the diagram indicates that activities B and C can be started and performed concurrently. Similarly, the diagram indicates that both activities B and C must be completed before activity D (high level design) can be started. Next, the diagram indicates that the activities E, F, and G cannot begin until D is completed. Finally, the project cannot be system tested (I) until both activities G and H have been completed.

The network diagram provides a picture of the sequence of activities required to complete the project. As shown in Figure 13, six different paths (sequences of activities) can be observed. One of these paths, for example, is A-C-D-F-H-I. Within a given sequence, e.g., A-C-D-F-H-I, each activity must be completed in turn before the following activity can be started. Note that each of these six activity sequences (paths) can be performed concurrently. The minimum project duration (make-span) will, therefore, equal the *longest* of the times required to complete the activity sequences.

The six paths and their associated times are:

Path	Weeks
A-C-D-F-H-I	16
A-C-D-E-H-I	17
A-C-D-G-I	11
A-B-D-G-I	12
A-B-D-E-H-I	18
A-B-D-F-H-I	17

Since the activities on path A-B-D-E-H-I require the longest overall time of 18 weeks, this establishes the minimum project length.

4.4.2 Properties of the Graph

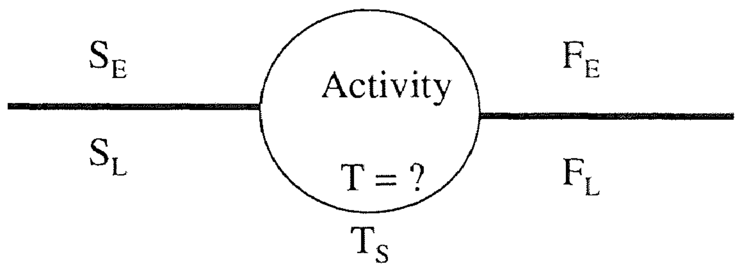
Analysis of the graph (Figure 13) obtained thus far reveals a number of properties which are important to project scheduling.

Let's consider the questions of how long the project is expected to take and when each activity may be scheduled. All basic scheduling computations first involves a forward and then a backward pass through the network. Based on a specified project start time, the *forward pass* computations proceed sequentially from the beginning to the end of the project giving the earliest start and finish times for each activity.

By the specification of the latest allowable occurrence time for the completion of the project, the *backward pass* computations also proceed sequentially from the end to the beginning of the project. They give the latest allowable start and finish times for each activity. After the forward and backward pass computations are completed, the *float* (slack) can be computed for each activity, and the critical paths are then identified.

4.4.3 Early-start Schedule and Forward Pass Calculation

The *early-start* schedule is developed by making a *forward pass calculation* through the network diagram, taking into account the required time for each project activity. The early-start time (S_E) and the early finish time (F_E) are determined for each activity and noted on the network diagram, using the conventions shown in Figure 14.



where:

- S_E = Activity early-start time
- F_E = Activity early-finish time
- S_L = Activity late-start time
- F_L = Activity late-finish time
- T = Activity time
- T_S = Activity total slack time

Figure 14 Symbol used in the network.

There are four rules for calculating the early start and finish times for each activity:

1. The early-start time for the initial activity in the network is set to zero, i.e., S_E = 0 for the START activity.

2. An activity can begin as soon as its preceding activity has been completed, i.e., S_E = the F_E for the preceding activity.
3. The activity early-finish time equals the early-start time plus the activity time, i.e., $F_E = S_E + T$.
4. When an activity or circle on the network diagram has more than one predecessor activity, i.e., more than one arrow entering the node, the activity early-start time equals the largest early-finish time of the preceding activities, i.e., S_E = largest of (F_{E1}, F_{E2}, ... F_{En}) for an activity having n predecessors.

The determination of the early-start and early-finish times for a project using these four rules, is illustrated with the numbers on top of the nodes (Figure 15). Since the initial activity (A) is the starting activity for the project, its early-start time (S_E) is set to zero. The activity requires three weeks to complete, so its early-finish time (F_E) equals three. Therefore, the early-start time for activity B equals three, and since activity B requires two weeks to complete, its early-finish times for these two activities provides an illustration of rules 1 though 3. The application of rules 1 through 4 to the activities in Figure 15 provides the early-start (S_E) and early-finish times (F_E) for the whole project. When both the activity times and the precedence relationships between the project activities have been considered, it appears that this project cannot be completed earlier than the end of week 18.

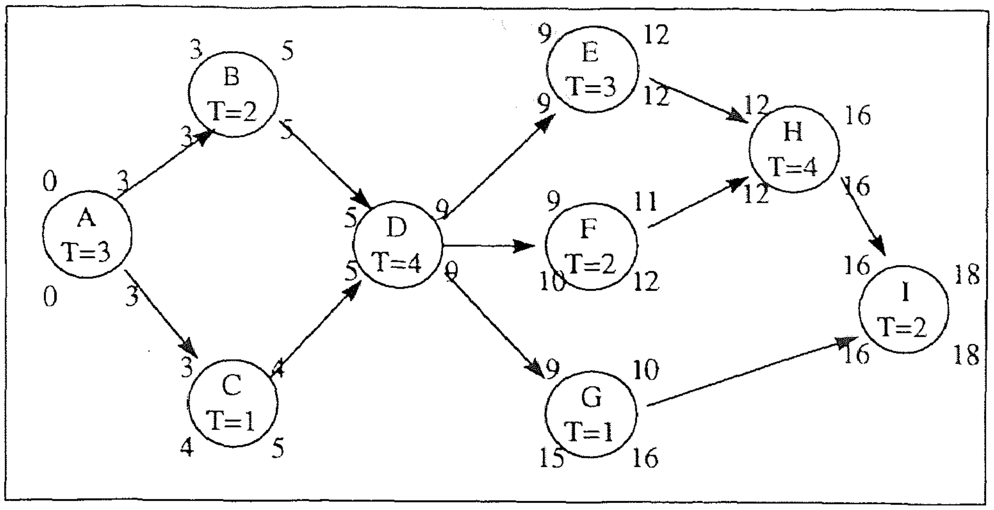


Figure 15 Early-start and late-start activities.

4.4.4 Late-start Schedule and Backward Pass Calculation

An alternative schedule can be constructed for the project in which the activities are scheduled as late as possible to meet the earliest project completion date. This schedule, called the late-start and finish schedule, helps to define managerial flexibility in scheduling individual project activities.

The late-start schedule is prepared by making a *backward pass calculation* through the network diagram, beginning with a stated project completion date for the last project activity, and working backward toward the first activity in the project. This scheduling procedure produces a late-start (S_L) and a late-finish time (F_L) for each activity. These times are usually noted under the nodes on the network diagram using the conventions shown previously in Figure 14. There are four more rules for calculating the late-start and late-finish times for each activity:

1. The late-finish time for the final activity in the network is set to the earliest project completion date, i.e., F_L = the F_E for the final activity.

2. The late-finish time for an activity equals the late-start time for the activity immediately succeeding it, i.e., F_L = the S_L for the succeeding activity.
3. The activity late-start time equals the late-finish time minus the activity time, i.e., $S_L = F_L - T$.
4. When an activity has more than one successor activities, i.e., more than one arrow leaving the node, the late-finish time for that activity is the smallest of the late-start times for those activities immediately succeeding that activity, i.e., $F_L = \text{MIN}(S_{L1}, S_{L2}, \ldots S_{Ln})$ for an activity having n successors.

4.4.5 Float Time

Once the early-start and late-start times have been determined for all of the activities in a project network, the total *float* time, i.e. the *slack time*, for each activity in the project can be determined. The total float time for an activity (T_F) is defined as the difference between its late- and early-start times ($T_F = S_L - S_E$). Alternatively, the total float (T_F) for an activity can be calculated as the difference between the early- and late-finish times (i.e., $T_F = F_L - F_E$). The total float time for each activity in our example graph is shown in Figure 16, and the detailed calculations are given in Table 2.

Table 2 Float time calculation.

Activity	S_E	S_L	T_F
A	0	0	0
B	3	3	0
C	3	4	1
D	5	5	0
E	9	9	0
F	9	10	1
G	9	15	6
H	12	12	0
I	16	16	0

Note that activity G can be started as early as the end of week 9 at the early-start time (S_E) and as late as the end of week 15 (S_L) without affecting the project completion date of the end of week 18. Therefore, the total float time (T_F) for activity G is 15 - 9, or 6 weeks. The total float time measure indicates the degree of flexibility that management has in scheduling the start time of a particular activity to best utilize the available resources or to respond to unexpected conditions.

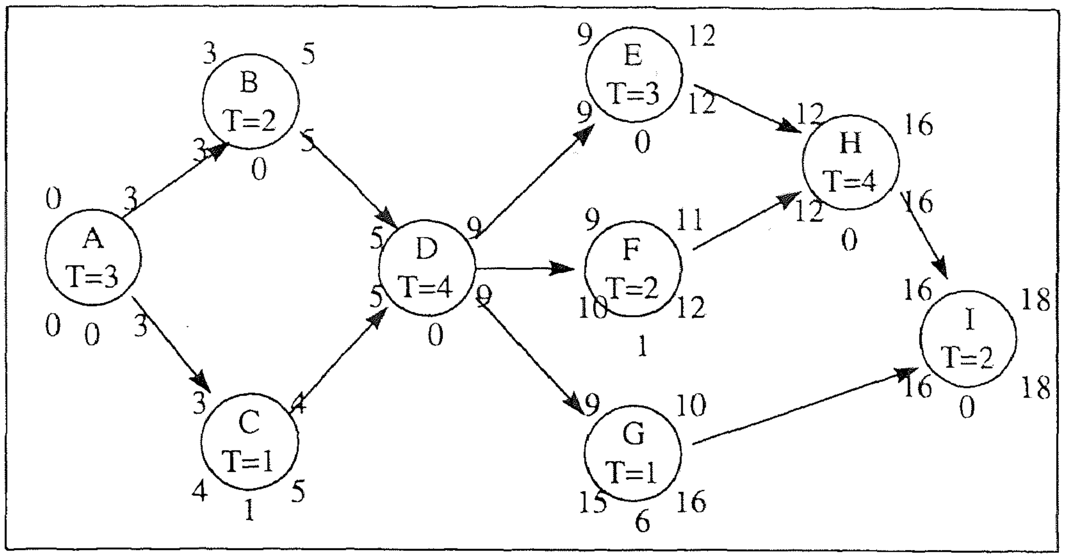


Figure 16 Float time in the graph.

There are two types of float, *path float* and *activity float.* Path float, as the name implies, is the total float associated with a path. For a particular path activity, it is equal to the difference between its earliest and latest allowable start or finish times. The path float denotes the amount of time (e.g. the number of working days) by which the actual completion time of an activity on the path in question can exceed its earliest completion time without affecting the earliest start occurrence time of any activity on the network critical path.

Activity float, also known as *free float,* is equal to the earliest start time of the activity's successor activity(s) minus the earliest finish time for the activity in question. Activity float indicates the amount of time that the activity can be delayed without affecting the earliest start of any other activity in the network.

Path float and activity float are useful indicators for risk management.

4.4.6 Critical Path and Critical Task

Several of the activities shown in Figure 16 have a total float time of zero, indicating that management has no flexibility in scheduling these activities. For example, since the early-start time for activity B equals its late-start time, this activity must be started at the end of week 3 or the completion of the entire project will be delayed beyond the end of week 18. Similarly, activities A, D, E, H, and I all have a total float time of zero.

The fact that activities A, B, D, E, H, and I have a zero total float time, and must be performed in the sequence shown in Figure 16, means that they form a *critical path* in completing the project. A delay in starting any of the activities on the critical path in the project network means that the project cannot be completed at the end of week 18. The determination of the critical path in a project network enables management to focus attention on those activities which are most crucial in completing the project on time. Other activities which do not lie on the critical path, do not warrant the same degree of managerial attention since there is some flexibility (as indicated by the total float times) in both starting and completing these activities.

It is possible for there to be more than one critical path in a project network. As an example, suppose that activity C required two weeks to complete instead of one. In this case, the project network would have two critical paths, A-C-D-E-H-I and A-B-D-E-H-I, both requiring 18 weeks to complete. In this case, activity C would have an activity total float time of zero, and the timely completion of this activity would also be of major concern to management. The determination of the critical path is a key element in ensuring the on-time completion of a project, and provides management with an important tool for identifying those project activities that deserve special attention in managing the project.

Activity D in Figure 16 is a *critical task*. Tasks E, F and G all depend on the completion of this critical task. Its delay would cause delay of all the tasks depending on it, and therefore, demands special attention.

Unlike the critical path, however, the definition of critical task is of semantic significance only. Another word, whether a task is considered as a critical task and the degree of its criticality are interpreted based on the depending tasks and the operating semantics. One can say, for example, that activity A is also a critical task, since both activity B and C are dependent on A's completion. But semantically, this may not be the case. To a certain degree, we can also regard each of the tasks on a critical path as a critical task, regardless whether there are more than one task waiting on its completion.

4.5 Scheduling with Resource Constraints

The forward and backward pass calculations discussed before produce schedules based on the assumption that no resource limitations are imposed. The only considerations were the precedence relationship and the duration of each project task. This *time-only* scheme, of course, is not realistic in practical situations.

This section discusses scheduling issues with resource constraints as additional consideration.

When resource constraints are put into consideration, the longest sequence of activities through the project may not be the same critical path anymore. It should be noted that with the basic time-only procedures there is one unique early-start time schedule, while under resource constraints many different early-start schedules may exist.

To understand these differences it is necessary to look at how limited resources affect graph float (slack time in the schedule).

4.5.1 Resources Limitation and Float

Figure 17 shows a simple CPM graph. The number under each node is the estimated task execution duration. Figure 18 shows the all early start bar-chart schedule for this network.

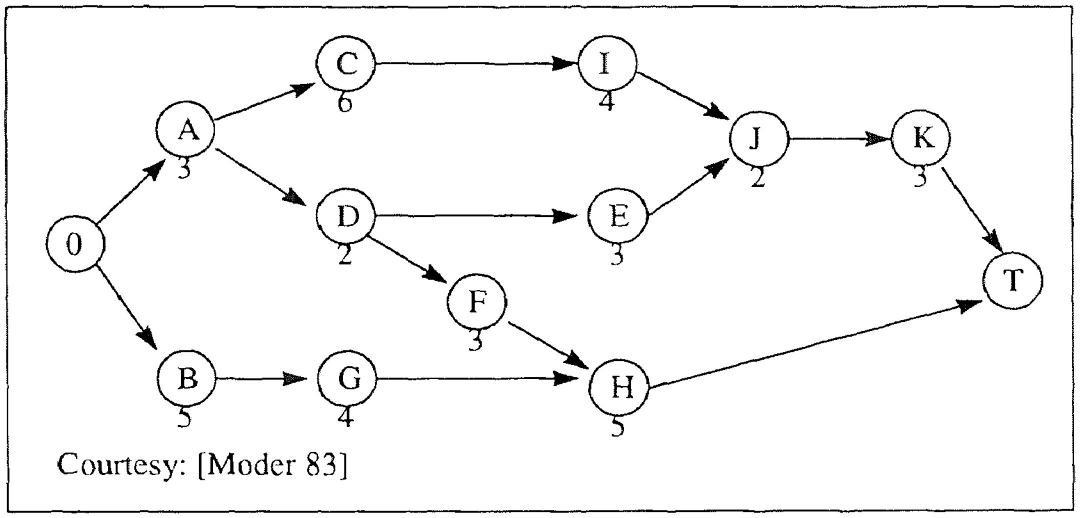


Figure 17 Sample graph with estimated duration.

As can be seen, the project duration is 18 weeks, the critical path is the activity sequence A-C-I-J-K, and tasks B, D, E, F, G, and H all have positive float shown with dashed lines. These float times are calculated by taking the difference between the forward pass schedule and the backward pass scheduled as discussed earlier.

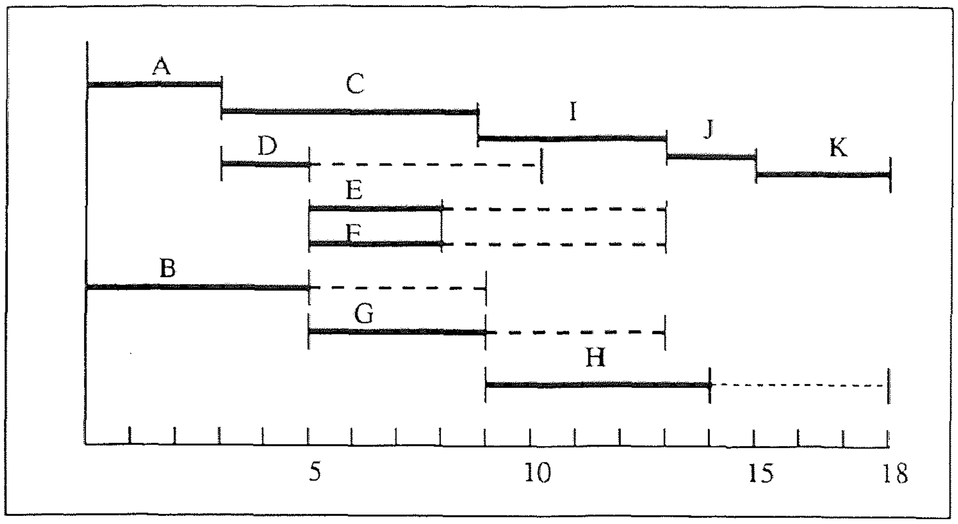


Figure 18 All early start schedule with unlimited resources.

Now assume that tasks C and G must be performed by the same person and the performance of these tasks cannot be done simultaneously. Assume also that tasks E and F have the same restriction, i.e. they are performed by the same person in a sequential order.

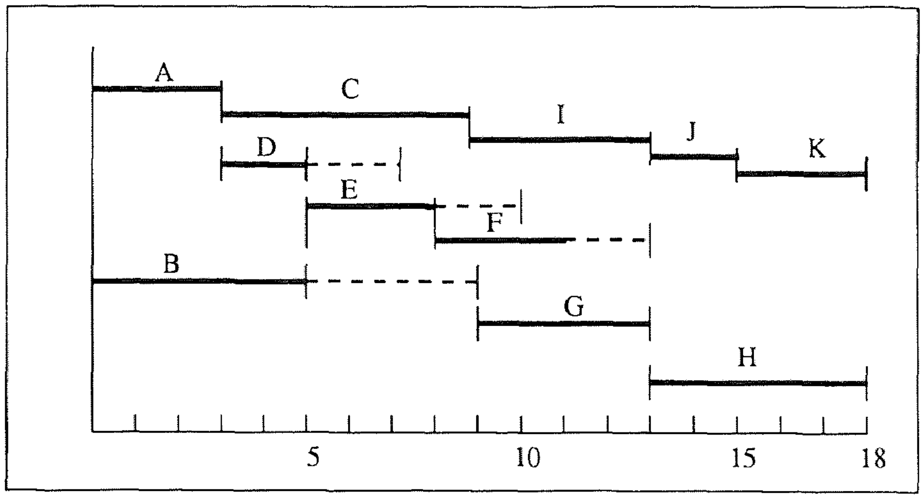


Figure 19 All early start schedule with restricted resources.

The direct result of these resource constraints is that neither tasks C and G nor tasks E and F can be performed simultaneously as indicated by the ES time-only schedule. One or the other of the tasks in each pair must be given priority and each pair must be sequenced so there is no overlap, as shown in Figure 19.

Examination of Figure 19 shows that, when resources for activities C/G and E/F, respectively, are constrained, the following is apparent:

- Activity G and H become critical, with slack reduced to zero.
- Activity D,E, and F have their slack reduced significantly.
- With activity E given priority over F as shown, the slack of tasks D and E become dependent upon F.
- No task can start earlier than shown, given the precedence relations and resource constraints, so this represents an early start schedule. However this schedule is not unique (as is the case with unlimited resources), since task F could have been sequenced before task E in resolving the resource conflict. In that case, the resulting schedule, though only slightly different from the one shown here, would be another ES schedule for the resource constraint case.

As this example shows, float can be affected in significant ways when resources are limited. In general, the following is true:

- Resource constraints reduce the total amount of schedule float.
- Float depends both upon activity precedence relationships and resource limitations.
- The early and late start schedules are typically not unique. This means that float values are not unique. These values depend upon the scheduling rules used for resolving resource conflicts.
- The critical path in a resource constrained schedule may not be the same continuous chain(s) of tasks as occurring in the unlimited resources schedule. A continuous

chain of zero-float tasks may exist, but since task start times are constrained by resource availability as well as precedence relations this chain may contain different tasks.

4.5.2 Scheduling for More Than One Project

The impact of resource constraints illustrated by the single-project example is magnified in scheduling multiple projects, i.e., situations where several separate, independent projects are lined together through their dependency upon a set of common resources. Figure 20 shows a hypothetical 3-project scheduling situation involving only 3 people. To further simplify the example imagine that tasks requiring a resource (person) use only one unit of his/her time. All tasks are shown on their respective resource bar charts at their early start times.

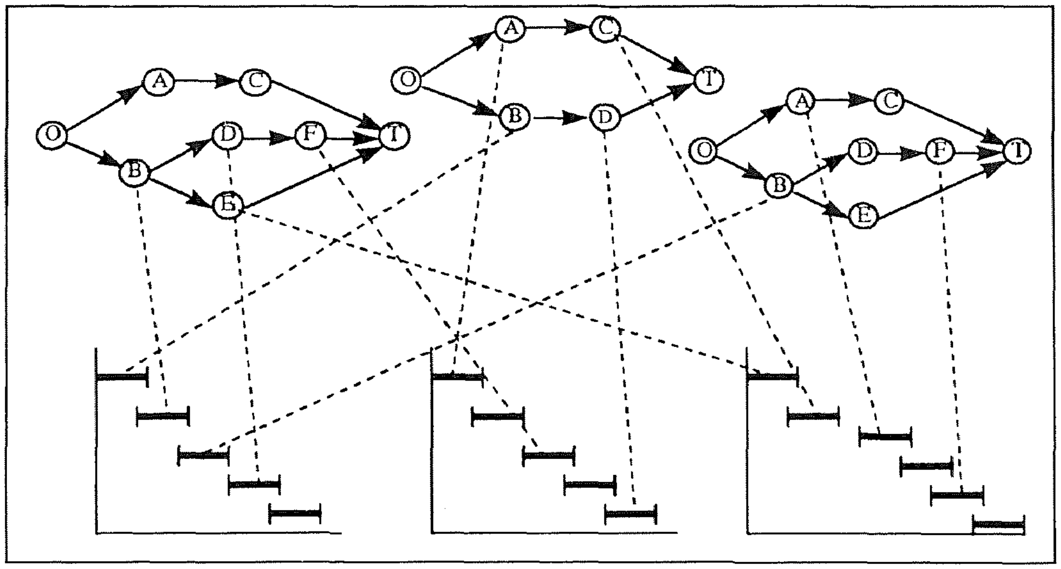


Figure 20 Multi-project scheduling interactions.

Analysis of these graphs reveals the domino-like series of events that might occur (depending upon activity float, and project finish times) as a result of delaying tasks to resolve particular resource conflicts. For example, delaying task B of project 1 (to resolve the conflict with task B of project 2) might cause the following:

- Delays in successor tasks D, E, and F of project 1.
- As a result, the creation of additional resource conflicts among tasks requiring persons 2 and 3 (which must be resolved).
- Therefore, additional delays in projects 2 and 3, and possibly even project 1 again.

4.5.3 Resource Loading Diagrams

Figure 21 shows the same network as Figure 17 with manpower requirements indicated above each task. By utilizing these resource requirements in conjunction with both an early-start schedule (such as shown in Figure 18 and a late-start schedule (not shown) the profiles of resource usage over time as shown in Figure 22 are obtained. These profiles are commonly called *resource loading diagrams*. Such diagrams highlight the period-by-period resource implications of a particular project schedule and provide the basis for improved scheduling decisions.

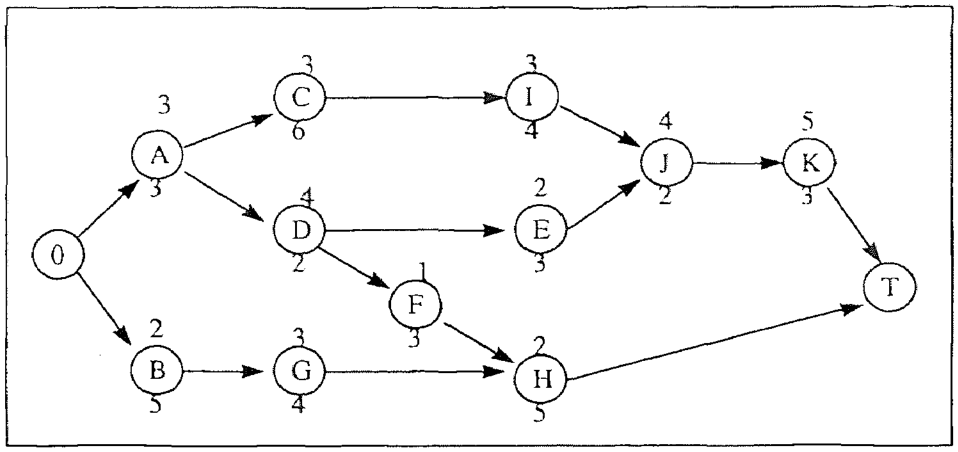


Figure 21 Sample graph with resource requirement.

Note, if number of existing resources are fixed, resource capacity usage can be calculated directly from the resource loading diagrams.

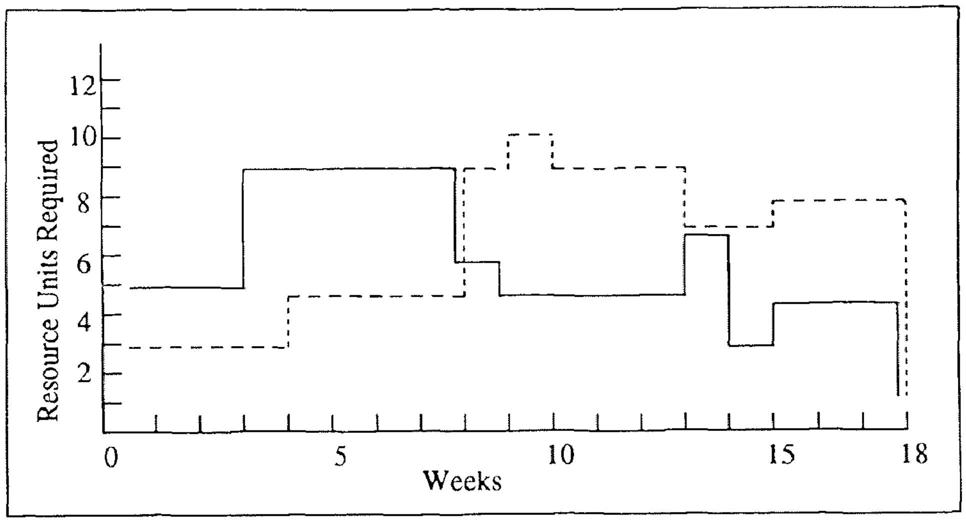


Figure 22 Sample graph with resource requirement.

Table 3 lists the period-by-period total requirements of man power (units of resource) for the graph described in Figure 21. Both the early-start and last-start schedules are shown. The period totals were used in constructing the resource loading diagram shown in Figure 22. Also shown in the table are the cumulative requirements for each period. These latter data can be usefully displayed in the form of *cumulative resource requirement* curves as shown in Figure 23.

Table 3 Early start (ES) and late start (LS) schedule requirements of the resource.

Period	ES Schedule Total Units	ES Schedule Cum. Units	LS Schedule Total Units	LS Schedule Cum. Units
1	5	5	3	3
2	5	10	3	6
3	5	15	3	9
4	9	24	3	12
5	9	33	5	17
6	9	42	5	22
7	9	51	5	27
8	9	60	5	32
9	6	66	9	41
10	5	76	10	51
11	5	76	9	60
12	5	81	9	69
13	5	86	9	78
14	6	92	6	84
15	4	96	6	90
16	5	101	7	97
17	5	106	7	104
18	5	111	7	111

The cumulative requirements curves can be very useful during the project planning stage for resource requirements and during the project execution stage in monitoring resource utilization. For example, as time progresses after the project has started, the cumulative resources required should lie within the closed area bounded by the early-start and late-start cumulative curves. If actual cumulative resources fall under the late-start curve the project is either behind schedule or the resource requirements were overestimated. Conversely, if they exceed the early-start curve the project is either ahead of schedule or the resource requirements were underestimated.

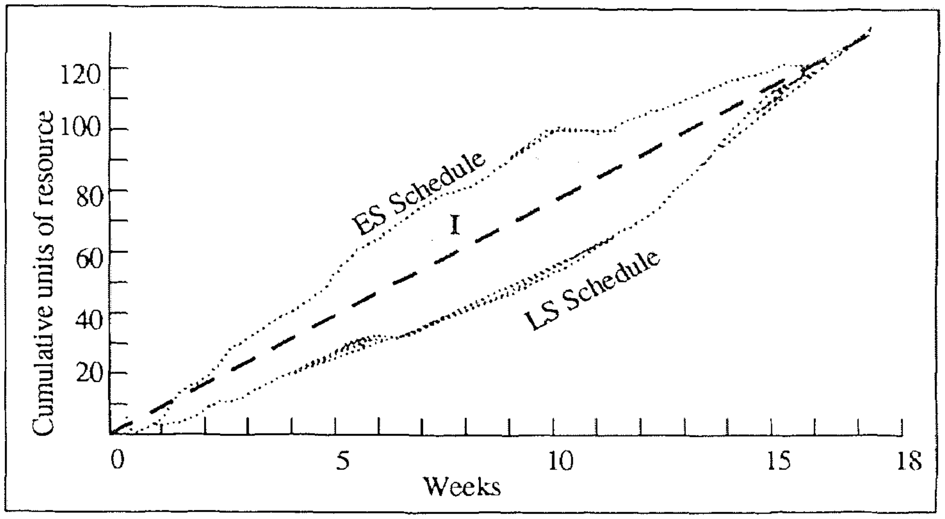


Figure 23 Cumulative resource requirements.

4.5.4 Resource Planning Using Cumulative Curves

One important use of the cumulative curves is in preliminary resource allocation planning. The magnitude of the total cumulative requirements and the slope of the average requirements line drawn in the center can be used to develop rough indicators of resource constraint "criticality" and of the likelihood of project delay beyond the initial forward-pass determined critical path duration. For example, line I in Figure 23 indicates the average weekly requirements for the manpower as:

average requirement = 111 total / 18 weeks = 6.2 people / week

4.5.5 Criticality Index

Suppose the manpower is available at a maximum level of 7 per week, a total of 126 could be utilized over the 18-week critical path duration, which is considerably more than the 111 required over that period. Thus, there is unlikely to be a project delay beyond the 18-week duration because of constraints on resources. This conclusion can also be drawn from the ratio of resources required to resources available, which is a rough measure of resource "tightness," or *criticality*. That is,

criticality index = avg. weekly manpower required / max. amount available weekly

= 6.2 / 7.0 = 0.88

The situation above can be contrasted to the case where only 6 manpower are available each week. In this case,

criticality index = 6.2 / 6.0 = 1.03

In 18 weeks a total of only 108 manpower will have been available, leaving some work unfinished and thus requiring an extension of the project beyond 18 weeks.

In general, higher values of the resource criticality index calculated here are associated with the most critical (i.e., most tightly constrained) resources.

Note that when discussing manpower, we can't ignore the fact that certain administrative overhead is necessary. For example, a dedicated person for project management and process control is essential in the development of the projects. However, this person may not be counted in the calculation of resource requirement discussed above. The semantic implication of manpower requirement is beyond the scope of this technical research.

4.6 Resource Constrained Scheduling Algorithms

There are two types of problems that need to be addressed in scheduling with resource constraints. One type of problem is a *soft resource limitation*. It occurs when sufficient total resources are available, and the project must be completed by a specified due date, but it is desirable or necessary to reduce the amount of variability in the pattern of resource usage over the project duration in order to reduce resource usage "spikes", during which time a considerable amount of resources are needed compared to the normal resource level. Thus, the objective of the scheduling algorithms is to level, as much as possible, the demand for each specific resource during the life of the project. This is called *resource leveling*. Project duration is not allowed to increase in this case.

Another type of problem is a *hard resource limitation,* also known as, *fixed resource limits*. It arises when there are definite limitations on the amount of resources available to carry out the projects under consideration. The scheduling objective in this case is to meet project due dates, insofar as possible, subject to the fixed limits on resource availability. Thus project duration may increase beyond the initial duration determined by the usual time-only calculations. The scheduling objective is equivalent to minimizing the duration of the projects being scheduled, subject to stated constraints on available resources.

The task of scheduling a set of project activities such that both precedence relationships and constraints on resources are satisfied is not an easy one, even for projects of only modest size. The basic general approach followed in both resource leveling and fixed resource limits scheduling is similar: set task priorities according to some criteria and then schedule tasks in the order determined as soon as their predecessors are competed and adequate resources are available.

4.6.1 Resource Leveling

The objective of resource leveling is to provide a means of distributing resource usage over time to minimize the period-by-period variations in manpower or other resources need. The essential idea is to schedule tasks within the limits of available float (slack) to achieve better distribution of resource usage. The float available in each activity is determined from the basic scheduling computations, without consideration of resource requirements or availability. Then, during the rescheduling, or "juggling" of tasks to smooth resources, project duration is not allowed to increase.

Measuring the relative *effectiveness* of the different schedules obtained by resource leveling was studied by Burgess as early as in the sixties [Burgess 62]. This method utilized a simple measure of effectiveness given by the sum of the squares of the resource requirements for each period (such as week) in the project schedule. It has been shown that, while the sum of the weekly resource requirements over the project duration is constant for all complete schedules, the sum of the squares of the weekly requirements decreases as the peaks and valleys are leveled. Further, this sum reaches a minimum for a schedule that is level (or as level as it can get) for the project in question.

4.6.2 Scheduling for Fixed Resource Constraints

The fixed resource constraint scheduling, also known as *constrained-resource scheduling*, or *limited resource allocation*, are techniques designed to produce schedules that will not require more resources than are available in any given period, with project duration which are increased beyond the original critical path length as little as possible.

One attempt in solving this type of problems is using optimization procedures for producing optimal solutions. These procedures can be divided into two categories: procedures based on *linear programming* and procedures based on *enumerative* and other mathematical techniques, such as "*branch and bound*" procedures. However, these procedures have not been proved effective for large and complex problems [Applegate91]. More practical procedures are still based on heuristic techniques [Moder83] which can be useful for multiple project problems as well as single projects. The schedules produced by these heuristic procedures may not be the theoretically best possible, but they are usually good enough to use for planning purposes in view of the uncertainties typically associated with activity duration and resource constraints and requirements.

4.6.3 Heuristic Algorithms

For any given problem, a large number of possible combinations of activity start times may exist, with each combination representing a different project schedule. This becomes a combinatorial problem: even for fairly small problems of 20 to 30 tasks the number of combinations is extremely large as to prohibit enumeration of all alternatives. One approach to solve these combinatorial problems is to use heuristic approach - a rule of thumb and a simple guide in problem-solving situations - to reduce the amount of effort required in coming up with an accurate solution. Many simple algorithms, such as "SPT" or "minimum float first", can be classified in this category.

4.6.4 Efficiency of Task Scheduling and Resource Usage

Once a graphical network has been set up as discussed earlier, measuring the differences between various graphs or impact of changes to a graph becomes relatively easy. This research has set up a framework that makes this measurement possible, and the actual measurement can be calculated and displayed as the graphs are manipulated.

The measurement can be classified into two categories: measures that indicate time characteristics of the graph, and measures that characterize resource demands/ availability. Examples of the measurements are shown below. In the actual implementation of a process modeling environment (see Chapter 6), the user should have the capability of disabling some of these measurements in order to concentrate on those measurements of interest towards the specific modeling objectives.

Measures that indicate time characteristics of the graph:

Sum of activity duration

Average activity duration

Variance in activity duration

Critical path duration

Total network float

Density: Sum of activity duration / (Sum of duration + Total free float)

Measures that characterize resource demands/availability

Cumulative resource requirements

Average resource requirement per activity

Average resource requirement per period

Resource utilization criticality index

The above measurements provide insights on how the tasks are being scheduled, and how the resources are being utilized. They provide the basis for comparison when comparing one graph to another, or to a higher level, one process model to another.

CHAPTER 5

MEASUREMENT OF STOCHASTIC MODELS

Software development environments in real life are subject to many sources of uncertainty. Among the sources with major impact are schedule delays, unexpected releases of high priority tasks such as bug fixes, unavailability of developers due to human related reasons, such as sick days, vacations, change of responsibilities, change of jobs, and so on. Another source of uncertainty is processing times, which may not be accurately estimated in advance. Thus, a good model of a software process would need to address these forms of uncertainty.

In this chapter, stochastic models are briefly examined in theory followed by a description of the statistical PERT approach. Applying the PERT approach in obtaining measurements during the process model manipulations enables us to deal with the problem of uncertainty effectively.

5.1 Scheduling Theory for Stochastic Models

In what follows, it is assumed that the *distribution* of the processing times, release dates, and due dates are all known in advance, that is, at time 0. The actual *outcome* or *realization* of a random processing time only becomes known upon the completion of the processing; the realization of a release date or due date becomes known only at the point at which it actually occurs.

In this chapter, the following notation is adopted. Random variables are capitalized, and the actual realized values are in lower case. Task j has the following quantities of interest associated with it.

X_{ij} = the random processing time of task j by resource i; if task j is only to be processed by one resource, or if it has the same processing times on each of the resources it may visit, the subscript i is omitted.

$1/\lambda_{ij}$ = the mean or expected value of the random variable X_{ij}.

R_j = the random release date of task j.

D_j = the random due date of task j.

w_j = the weight (or important factor) of task j.

A random variable from a continuous time distribution may assume any real non-negative value within one or more intervals. The distribution function of a continuous time distribution is denoted by F(t) and its density function by f(t), that is,

$F(t) = P(X < t) = \int^{t} f(t)\, dt$, where

$f(t) = dF(t) / dt$

provided the derivative exists.

An important example of a continuous time distribution is the *exponential distribution*. The density function of an exponentially distributed random variable X is

$f(t) = \lambda e^{-\lambda t}$,

and the corresponding distribution function is

$F(t) = 1 - e^{-\lambda t}$,

which is equal to the probability that X is smaller than t. The mean or expected value of X is

$$E(X) = \int^{\infty} tf(t)dt = \int^{\infty} tdF(t) = 1/\lambda.$$

The parameter λ is called the rate of the exponential distribution.

Often in stochastic scheduling, two independent random variables have to be compared with one another. These comparisons are based on properties referred to as *stochastic dominance*, that is, a random variable *dominates* another with respect to some stochastic property.

The random variable X_1 is said to be *larger in expectation* than the random variable X_2 if

$$E(X_1) \geq E(X_2).$$

The random variable X_1 is said to be *stochastically larger* than the random variable X_2 if

$$P(X_1 > t) \geq P(X_2 > t) \quad \text{or}$$

$$1 - F_1(t) \geq 1 - F_2(t)$$

for all t. This ordering is usually referred to as stochastic ordering and is denoted by

$$X_1 \geq_{st} X_2.$$

During the evolution of a stochastic process, new information becomes available continuously. Task completion and occurrences of random release dates and due dates represent additional information that the decision maker may wish to take into account when scheduling the remaining part of the process. The amount of freedom the decision maker has in using this additional information is the basis for the various classes of decision making policies. The following classes are defined:

1. Under a non-preemptive static list policy, the decision maker orders the tasks at time 0 according to a priority list. This priority list does not change during the evolution of the process, and every time a resource is freed the next task on the list is selected for processing.

2. Under a preemptive static list policy the decision maker orders the tasks at time 0 according to a priority list. This ordering includes tasks with nonzero release dates, that is, tasks that are to be released later. This priority list does not change during the evolution of the process, and at any time the task at the top of the list of available ιasks is the one to be performed by the resource.

Under this class of policies the following may occur. When there is a task release at some point ant the task released is higher on the static list than the task currently being processed, then the task being processed is preempted and the task released is put in the system.

3. Under a non-preemptive dynamic policy, every time a resource is freed, the decision maker is allowed to determine which task goes next. The decision at such a time point may depend on all the information available, for example, the current time, the tasks waiting for processing, the tasks currently being processed on other resources, and the amount of processing these tasks already have received on these resources. However, the decision maker is not allowed to preempt; once a task begins processing, it has to be completed without interruption.

4. Under a preemptive dynamic policy, at any time the decision maker is allowed to select the tasks to be processed by the resources. The decision may depend on all information available and may require preemption.

There are several forms of optimization in stochastic scheduling. Whenever an objective function has to be optimized, it should be specified "in what sense" the objective is to be optimized. One form of optimization is in the *expectation sense*, for example, one wishes to minimize the expected make-span, that is, $E(C_{max})$ and find a policy under which the expected make-span is smaller than the expected make-span under any other policy. A stronger form of optimization is optimization in the *stochastic sense*. If a schedule or policy minimizes C_{max} stochastically, the make-span under the optimal schedule or policy is stochastically less than the make-span under any other schedule or policy. Stochastic optimization implies optimization in expectation.

Note, the use of the word *optimization* here does not necessarily mean finding an *optimal* point for a solution. It simply means an *improvement* in a loose sense.

Based on the classes of policies discussed above, measurements of different stochastic machine models can be obtained, these include stochastic flow shops, stochastic open shops, and stochastic job shops. The stochastic flow shop machine model is closer to the situation in a software development environment and can be studied further.

Consider two resources in series with unlimited storage between the resources and no blocking. There are n tasks. The processing time of task j by resource 1 is X_{1j}, exponentially distributed with rate λ_j. The processing time of task j by resource 2 is X_{2j}, exponentially distributed with rate μ_j. The objective is to find the non-preemptive static list policy, or permutation schedule, that minimizes the expected make-span $E(C_{max})$. Note that this problem is a stochastic counter-part of the deterministic problem $F2 \mid \mid C_{max}$ and can be solved based on the following theorem.

Theorem: Sequencing the tasks in decreasing order of λ_j - μ_j minimizes the expected make-span in the class of non-preemptive static list policies, the class of non-preemptive dynamic policies, and the class of preemptive dynamic policies.

5.2 PERT Statistics Analysis

One of the difficulties associated with stochastic scheduling models is the accuracy of task duration estimation. The PERT approach uses three different estimates for each task duration estimation, the optimistic time, most likely time, and pessimistic time. It is based on the assumption that the distribution of these task completion times is in *normal distribution*. Using a guideline of 5 and 95 percentile, the three estimated values can be defined as follows:

Optimistic Performance Time (a): This is the time which would be improved only one time in twenty if the activity could be performed repeatedly under the same essential conditions.

Most Likely Time (m): Also called the *modal value* of the distribution, this is the value which is likely to occur more often than any other value.

Pessimistic Performance Time (b): This is the time which would be exceeded only one time in twenty if the activity could be performed repeatedly under the same essential conditions.

5.2.1 Normal Distribution and Central Limit Theorem

Let t be the completion time of task j, for n tasks the arithmetic mean of these task duration values can be calculated as follows:

arithmetic mean: $\mu = (t_1 + t_2 + \ldots + t_n) / n$ (1)

This is the measurement of the central tendency. The measurement of the variability, or the standard deviation can be calculated as follows:

standard deviation: $\alpha = \{[(t_1 - \mu)^2 + (t_2 - \mu)^2 + \ldots + (t_n - \mu)^2] / n\}^{1/2}$ (2)

Suppose m independent tasks are to be performed in order. This can be the case when all the tasks lie on the critical path of a graph. Let $t_1, t_2, \ldots, t_m$ be the times actually required to complete these tasks. Let $t_{e1}, t_{e2}, \ldots, t_{em}$ be the means and $V_{t1}, V_{t2}, \ldots, V_{tm}$ be the variances. Now define T to be the sum:

$T = t_1 + t_2 + \ldots + t_m$

Note that T is also a random variable and thus has a distribution. The Central Limit Theorem states that if m is large, the distribution of T is approximately normal with mean E and variance V_T given by

$E = t_{e1} + t_{e2} + \ldots + t_{em}$

$V_T = V_{t1} + V_{t2} + \ldots + V_{tm}$

That is, the mean of the sum is the sum of the means; the variance of the sum is the sum of the variances; and the distribution of the sum of activity times will be normal regardless of the shape of the distribution of actual activity performance times.

5.2.2 Estimation of the Mean and Variance of the Task Performance Times

It is commonly known in statistics that for normal distributions the standard deviation can be estimated roughly as 1/6 of the range of the distribution. This follows from the fact that at leas 89 percent of any distribution lies within three standard deviations of the mean, and for the normal distribution this percentage is 99.7+ percent. Hence, we can use time estimates, a and b, to estimate the standard deviation $(V_t)^{1/2}$ or the variance, V_t, as follows:

$$(V_t)^{1/2} = (b - a) / 3.2 \qquad \text{or} \qquad V_t = [(b - a) / 3.2]^2 \qquad (3)$$

A simple formula for estimating the mean, t_e, of the activity time distribution has also been developed [Moder 83]. It is the simple weighted average of the estimates a, m, and b as follows:

$$\text{Mean: } t_e = (a + 4m + b) / 6 \qquad (4)$$

It should be pointed out that the mean is equal to the most likely or modal time (t_e = m), only if the optimistic and pessimistic times are symmetrically placed about the most likely time, i.e., only if b - m = m - a.

To illustrate the use of the statistical PERT approach, consider the graph shown in Figure 24. The three value estimate of the tasks are as follows:

Task:	a	b	c	d	e	f
a,m,b:	1,2,3	2,4,6	1,2,3	2,3,4	3,4,5	2,3,4

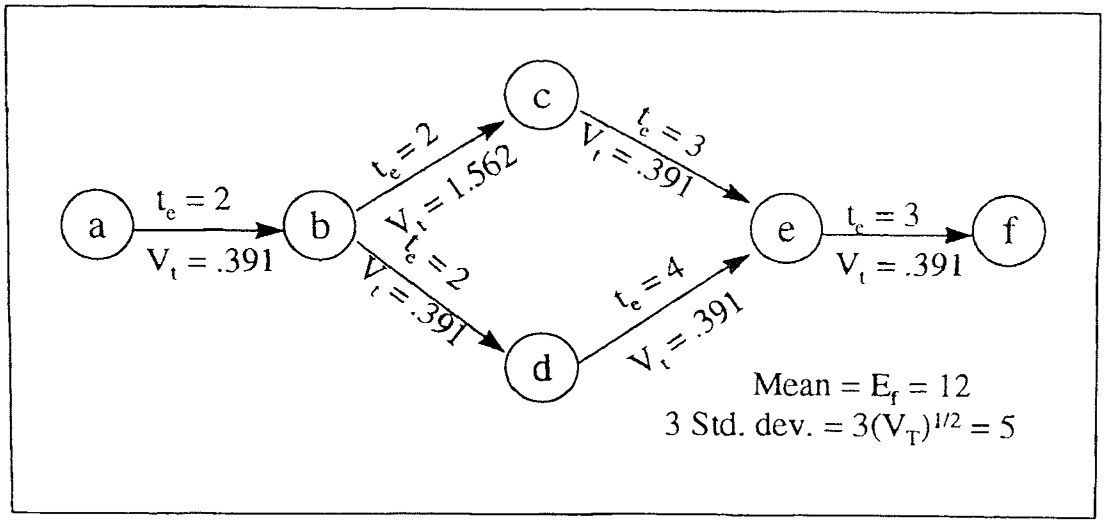


Figure 24 Basic graph with t_e and V_t for each activity.

The values of te, and V_t are computed according to equations (3) and (4). For example, for task a,

$t_e = (1 + 4 \times 2 + 3) / 6 = 2$

$V_t = [(3 - 1) / 3.2]^2 = 0.391$

The result of the forward pass are indicated by the time scale as shown in Figure 25.

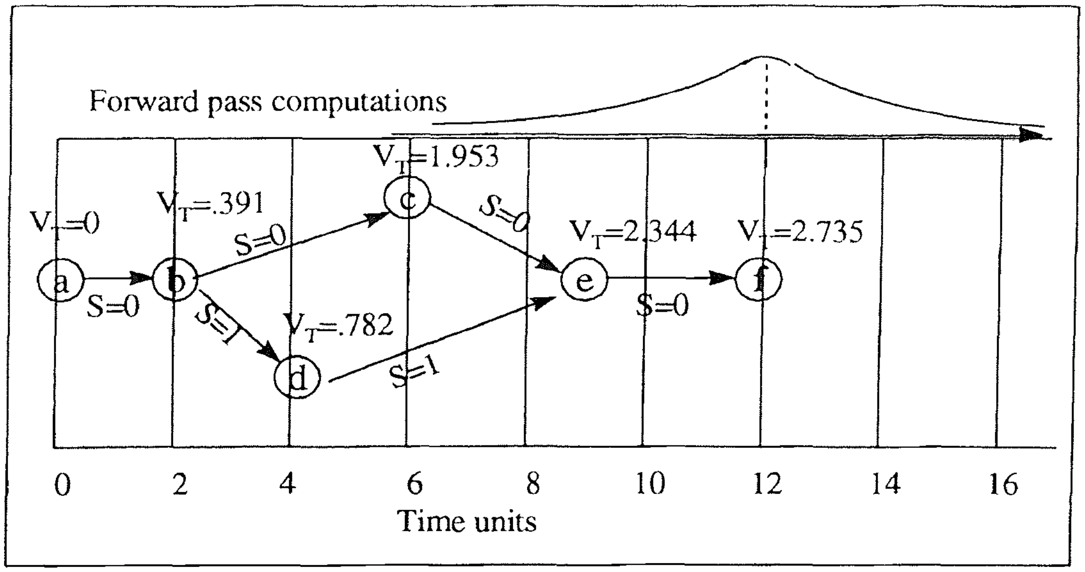


Figure 25 PERT statistical computations.

In addition to the above calculations, the statistical PERT approach also enables us to calculate the probability of meeting a scheduled target date (due date) for all tasks.

CHAPTER 6

AN INTEGRATED PROCESS MODELING ENVIRONMENT

The lack of industry-wide adaptation of process modeling as part of software development activities can be attributed, in part, to the lack of an environment where process modeling and software engineering activities are conveniently combined. Chapter 3 discusses the mechanisms that these integrated activities can be carried out. This chapter further describes the process modeling environment from the implementation's view point.

6.1 An Integrated Process Modeling Environment

An Integrated Process Modeling Environment (IPME) provides a base where process description, simulation and analysis, as well as software engineering activities can be

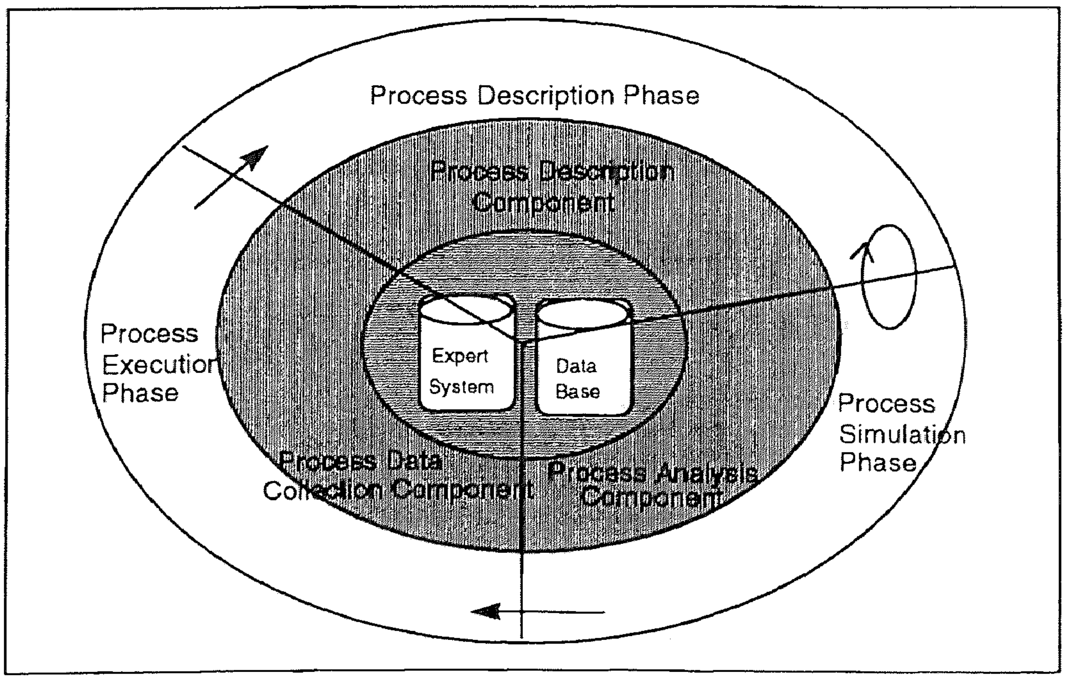

Process Description Phase
Process Description Component
Process Execution Phase
Expert System
Data Base
Process Simulation Phase
Process Data Collection Component
Process Analysis Component

Figure 26 An integrated process modeling environment.

carried out. It integrates databases, expert systems, tools, description graphs, and other process modeling and software engineering tools into one system.

An IPME consists of three major components (see Figure 26):

- Process Description Component (PDC)
- Process Analysis Component (PAC)
- Process Data Collection Component (PDCC)

Process modeling activities can be divided into three phases with each component above used primarily for one phase.

- Process Description Phase
- Process Simulation Phase
- Process Execution Phase.

During the Process Description Phase, the existing or the proposed process of the software development in the organization, along with the parameters and constraints affecting the process, are entered into the system using process description mechanisms and the tool sets provided by the PDC.

During the Process Simulation Phase, the process is simulated through interacting with the PAC. This Process Simulation Phase simulates real process execution in the Process Execution Phase. In this phase, the described process is analyzed according to the input data in the previous phase and the data stored in the process database. The analysis is carried out in a process engine which is equipped with a formal process description formalism. The output of this phase is a process outlook that contains analysis to the effectiveness of the process exhibited in terms of measurements. The process improvement personnel should be able to modify the process. If approved, the process database is automatically updated, and a set of new collectable items are entered into the database while obsolete items are being deleted from the system.

The heart of the PAC is a process modeling engine using graphs as its underlying modeling formalism, so that both the scheduling algorithms on graphs and algorithms in graph theory can be utilized for process manipulation. To enhance the intelligence of the process modeling engine, a rule-based process knowledge base can be added, which may be separated from the process database discussed above. This rule-based process knowledge base should be generated and modified automatically with no or little interference from knowledge engineers (except probably for the first time when the knowledge base is set up). The knowledge base is generated using the rules in the knowledge base and the data collected during the Process Execution Phase. The PAC of the IPME consists of a task analysis engine, an inference engine and a collection of process knowledge base, the expert system shells, and an expert system that can modify process rules based on the collected data and user input.

During the Process Execution Phase, the engineering development group conducts engineering activities according to the prescribed process. Process data collection is a major part of the activities in this stage. The PDCC of the IPME is to provide a set of tools and mechanism for collecting the data gathered during the execution of the process. The collected data forms a process database, which can be used by the other components. Central to the PDCC is a process monitoring function, which is used to analyze the environment changes and their impact to the current process.

In order to collect data, certain tools used for project development, such as load build tools, source code control libraries, and so on, can be integrated into the IPME so that the output from those tools can be directly used as input to the PDCC.

With the IPME, processes can be studied and their effectiveness can be analyzed. These analysis should be carried out throughout the life cycle of the process, from the process description stage to process execution. This ensures that the process to be taken is fully studied before it is put into action, and the process is fully monitored during its execution, and data collected are put back to the system to drive process modifications.

This feedback loop ensures that risk management is carried out throughout the process life cycle. During the Process Simulation Phase, the user entered data, the previous performance database, the process knowledge base are all activated in an attempt to identify risk areas and deficiencies. This is a deductive analysis of the process. During the Process Execution Phase, the IPME system monitors execution of the process through measurement and data collections. Risks are identified as soon as they are indicated by the collected data. The process improvement personnel can then go back to the Process Simulation Phase to put the newly derived process under study. Thus, data collected through process execution are used to drive process modifications, and the process can be modified with ease and on a timely basis.

After the Process Execution Phase, the collected data enables the IPME system to do a retrospective analysis to the process. This postmortem analysis and measurement can lead to improved process for the next process cycle.

6.2 Users' Interaction with IPME

The users of the IPME are typically the process improvement personnel, management team, and possibly, project management team as well.

Users' perspective to a process and a process model can be formulated into several sets of parameters interacting with each other. This is consistent with the perspective with machine models discussed in Chapter 4 and 5. On a user's level, these parameters include the following:

- a set of tasks (τ)
- a set of resources (ρ)
- a set of restrictions (γ)

A process, P, is then a function of all the sets interacting with each other:

$P = \{\tau, \rho, \gamma\}$.

The set of tasks, τ, are the building blocks of the whole process. The user defines the set of process tasks for the specific domain or applications. These tasks can be defined at different levels based on the level of organization that the process is to be modeled. Tasks can be broken down into smaller tasks through task refinement. This activity continues until a satisfied level of abstraction has been achieved for that particular process being modeled. Note that these tasks are process tasks onto which project tasks are being mapped. For details on process tasks, project tasks and their interactions, see Chapter 3.

The set of resources, ρ, include all the resources that affect the process. These include the set of people, the set of equipment (if applicable), the set of inputs such as test procedures, and the set of outputs such as source code and design documents. (To simplify the resource assignment in process modeling, the only resource type considered in this research is personnel. Other resource types, such as computing equipment and labs, are not considered.)

The set of restrictions, γ, define any restrictions imposed on the tasks and resources. Such restrictions may include scheduling restrictions (deadlines, milestones), resource restrictions (e.g. resources A and B are mutually exclusive), resource allocation restrictions, (e.g. person A cannot do task C), and so on.

The complexity of the process tasks interactions and the mechanisms of process representation should be transparent to users. The user interface of the IPME should take the advantage of the state of art graphical user interface capabilities on PC, workstations or other hardware platforms.

The PDC provides interfaces prompting users to enter all the data required by the process model. The user should be able to query the data entered, modify the data, experiment with the data, and so on. To enter data, the user should do the following (not necessarily in this order):

- Enter process tasks.
- Setup functional links for all the process tasks.
- Enter project tasks.
- Map project tasks to process tasks.
- Assign duration for project tasks.
- Assign resource to project tasks.
- Invoke the process modeling engine to validate the input and to resolve errors such as resource assignment conflicts, loops in dependency graphs, etc.

After data being entered, the PAC proceeds to analyze the input data, the stored database, the process knowledge base, the requirements and restrictions, etc. Based on this analysis, the system calculates measurement data, such as make-span, float, etc. and presents it to the user. This kicks off the user/system interactions. The user keeps modifying and experimenting with the data (tasks, resource allocations, relationships, restrictions, etc.) through the PDC, and the system keeps presenting new measurement data through the PAC. This cycle continues until the user is satisfied with the result.

In addition to the validation and measurement functions, the PAC is also capable of setting up a recommended process based on user input and existing data. If invoked, the system recommended process should be at least as good as the user defined process, since it can always go back to the user defined process if that process is proven to be a better one. In most cases, however, the set of tasks and resources are so large that coming up with an ideal process solution is beyond human being's normal capacity. This is where the process modeling tools can help. Even though in a simple situation when the user can define an ideal process solution, the process still needs to be recorded in the IPME system so that execution of the process can be monitored. When operational environment changes (tasks being delayed, resources not available, change of requirements, etc.) the process can be made adapt to the changes quickly.

There are many ways of arranging process tasks. The objective of the IPME system is to help obtain an improved process (within the constraints given by the user) in terms of the *net process time* (T_p) and the *resource capacity usage* (C_r). The objective of the user is to fine-tune the obtained process based on such human factors as policies, managerial aspects, motivations, and so on. These human factors can also be translated into process constraints or entered into the system as process parameters. The end result of this human-machine cooperation is a final process that is improved in the sense that it balances out both the optimal point as measured mathematically by the system (based on the entered or collected data) and the optimal point as judged semantically by the user based on the human factors.

6.3 Graphs in the Process Modeling Engine

As mentioned earlier graphs are being used in the process modeling engine for the PAC. Usage of graphs makes it convenient for studying the process in terms of scheduling theory and graph theory.

6.3.1 Task Representation

A task system can be represented in the form of a directed graph, $G = (V, E)$, where the set of vertices, V, represents the set of tasks, τ, and the set of edges, E, represents the set C (Figure 27). Thus, an edge, $e \in E$, represents a partial ordering of the two tasks T_1 and T_2 in τ that are connected by e.

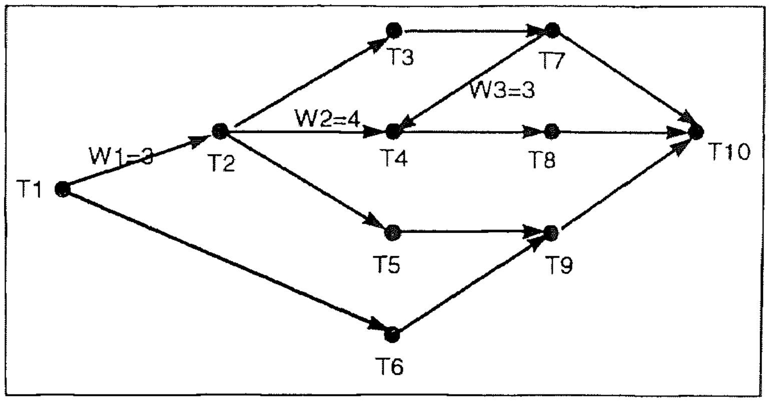


Figure 27 Functional dependencies of tasks.

Each edge, e, from vertices T_i to T_j can have an edge weight, w, associated with it. Depending on the layer of task representation (see the section, "Task Representation with Layered Graphs" for details), this weight can be used to represent different parameters. For example, if the task duration is the parameter that we are interested in, then we can use the weight of the edge, e, to represent the task duration of the vertex which the edge is incident from, i.e. vertex T_i.

Figure 27 is an initial and simplest representation of the set of tasks in τ. The directed edges show the interdependencies, or ordering, of the tasks. These ordering are due to functional dependencies, not resource dependencies. Resource dependencies can be represented with a similar, but separate, graph.

As shown in the figure, tasks t2 and t6 have dependency on task t1, task t9 has dependency on tasks t5 and t6, and so on. Tasks t3, t4 and t5 are tasks that can be done in parallel, so are t2 and t6. w1=3 on the edge connecting t1 and t2 shows that the duration of the task t1 is 3 units (weeks, months, or whatever other measurement unit is chosen).

This implies that the weight of the edge connecting t1 and t6 is also 3, since the duration of executing t1 is 3.

Each vertex in the graph has all the parameters associated with the task as identified in the task system. The set of parameters associated with a task will be expanded as needed.

There are many measurements that can be obtained. Two important ones are being used to assess the effectiveness of the process, *net process time* (T_p) and *resource capacity usage* (C_r). The net process time for the tasks, τ, is the duration between the start of the first task and the conclusion of the last task. This measurement can be used to anticipate how long execution of a particular task set would require from start to end. Translated in graphs terms, this is the *make-span* (C_{max}). The resource capacity usage is a percentage of the maximum capacity of each resource that is being used. This can be used to measure how effectively each resource is being used for a particular task set. Variations of this can be used to measure resource capacity usage for a particular resource type, groups of resources, or the whole resource set (ρ). In graphs terminology these are *floats* of the tasks or paths.

By representing tasks in the task system using graphs, tasks can be studied in terms of graph theories. Thus, the whole set of tasks can be rearranged according to the functional dependencies, the resource dependencies, as well as other restrictions. The objective is to be able to manipulate the graph so that a new graph, representing new and improved task execution orders, can be derived. This new ordering should show improvement over the initial ordering in terms of the net process time for τ and the resource capacity usage for the resources, ρ. The goal is to minimize the net process time and to maximize resource capacity usage.

6.3.2 Task Representation with Layered Graphs

The tasks as described by the user form a functional relationship, which can be represented by a graph. When resources are assigned to each individual task, the resources themselves form another relationship, which can be described by another graph. If we put the two graphs together, it becomes a two-layered graph (Figure 28). In Figure 28, one graph (the functional dependency graph) is represented in solid lines and the other graph (the resource dependency graph) is represented with the same set of solid lines plus the dashed lines. In the resource dependency graph, P1, P2, P3... represent persons 1, 2, 3, ... and so on.

Each task has a number of parameters and restrictions associated with it. Parameters of the same type form a special relationship which can be represented by a graph.

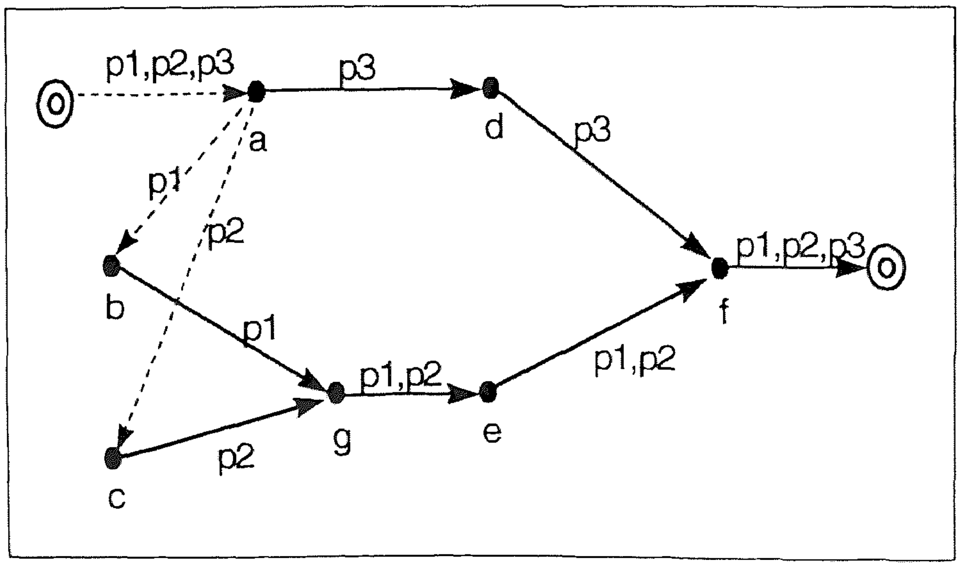


Figure 28 Layered graphs for the task set.

Therefore, there can be multiple layers of graphs associated with a set of tasks. Each layer represents one dimension of the process. This allows the process engine take different approaches or use different algorithms on different layers while maintaining the layer interactions by connecting them with the same set of vertices (tasks). For example, in Figure 28, one layer is for the functional dependency and the other layer is for resource dependency. In order to optimize a process, all layers should be considered. Another word, the optimal point should be a balanced point for all the layers.

6.3.3 A Process Instance Example

Figure 29 shows a simple process instance. In this directed graph, the vertices a to f are specific tasks, and the edges show the task ordering. The value on each edge show the personnel assignment, for example, p_1, p_2 and p_3 are allocated to do task a and p_3 is allocated to do task d when a is done.

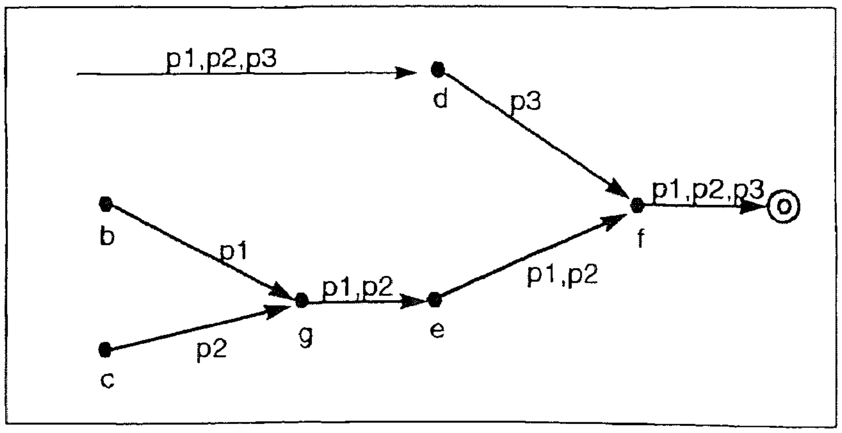


Figure 29 A process instance example.

Depending on the duration of the tasks, the resource allocation restrictions, and other process parameters, one way of allocating the personnel to different tasks is to let p_1 and p_2 finish task e and get ready for task f. Then let p_3 join p_1 and p_2 to start task a. An alternative of this is to let p_1 and p_2 finish task g, then let p_3 join p_1 and p_2 to start task a. At task d, p_1 and p_2 start working on task e while p_3 continue working on d.

Which is a better process? The process modeling tool and the user need to have a joint assessment on this.

One example of the user interface may be as follows:

Task Set: t_1, t_2, t_3, t_4...

Personnel: p_1, p_2, p_3, p_4...

t_1 p_1, p_2, p_3, p_4...

Select personnel(s):	<u>p1, p2</u>
Select duration:	<u>20 days</u>
Restrictions:	<u>none</u>

...

t_1 p_1, p_2, p_3, p_4...

Select personnel(s):	<u>p5</u>
Select duration:	<u>15 days</u>
Restrictions:	<u>none</u>

...

In this example, personnel assignment is done for each task. The user is given prompts to select the personnel who can do this task, and for this particular resource assignment, how long the task will take, etc. This is repeated for the same task with other alternatives. Note, this selection is the user's specification of who *can* do the task, not who *should* do the task. The user's preference may be entered as additional parameters.

The more options the user give, the more flexible the system can be when providing a recommended process. On the other hand, the more restrictions the user provide to the system when specifying the parameters, the more accurate the recommended process can be compared with the preferred user specifications.

6.4 Process Measurement and Improvement Through Graph Analysis

After tasks are decomposed, parameters and restrictions associated with each task are specified, and functional and resource dependencies are identified, the tasks are going through a series of interactive task manipulations to put the task in order, to assign resources for each task and to fine-tune the whole process by rearranging tasks or resource assignments or both.

6.4.1 Ordering of Tasks by Topological Sort

When tasks are identified and the partial ordering relationships are specified, the tasks need to be put in an execution order. The initial execution order can be done without regard to the resource allocations, i.e. the execution order is dominated only by the partial ordering relationship given by the user.

This execution order can be obtained by performing a topological sort on the tasks. After the sorting, it is very likely that several tasks can be started independently. This gives the system flexibility to decide when to start which task. The system only knows certain task cannot be started until certain other tasks are done. When resources are also allocated to the tasks, the system also has to know that certain task cannot be started until certain point in time, t_i, due to resources dependencies. Therefore, there are two dimensions in execution of tasks: ordering and timing. The task dependencies determine the task execution ordering. The resource dependencies determine the task execution timing. Together they define which task should be done at which time and by whom.

The end result of this ordering is a network of tasks in graphs with several independent paths from sources to sinks. There could be more than one ordering, resulting in more than one graphs with different paths. Overlaying all the graphs on top of each other (conceptually), the single graph will show all the possible resource assignments. Using "shortest path" algorithms, the system can find an optimal resource assignments and task ordering to present to the user. Again, this optimal ordering is measured in terms of the net process time and the resource capacity usage. This has been discussed in Chapter 4 in details.

Figure 29 shows one example of the task ordering after a topological sort. As shown in the figure, p_1, p_2 and p_3 execute task a. p_3 continues to execute task d. When p_1 and p_2 finish task e, p_3 joins them to finish task f.

The problem is that tasks a, b, and c cannot be started together, because p_1 and p_2 are involved with tasks b and c as well as task a, resulting in *resource contention*. To show the resource dependencies, the graph is converted to a resource dependency graph as shown in Figure 30.

In Figure 30, p_1, p_2 and p_3 start working on task a. Then p_1 starts working on task b while p_2 starts working on c. After that, both p_1 and p_2 work on task g. Meanwhile, p_3 is assigned to work on d.

Note, in the task dependency graph, the value on an edge is the person assigned to a task that the edge is incident from; in the resource dependency graph, the value on an edge is the person assigned to a task that the edge is incident to.

After the transformation of the graph into a resource dependency graph, the duration from start of task a to finish of task f can be traced. When one task has resource dependencies on more than one task, process slack time may result, since it may not happen that all the resources involved with the depended tasks get freed up at exactly the same time. In certain cases, the waiting task can be started with partial resources. In other cases, this is not possible.

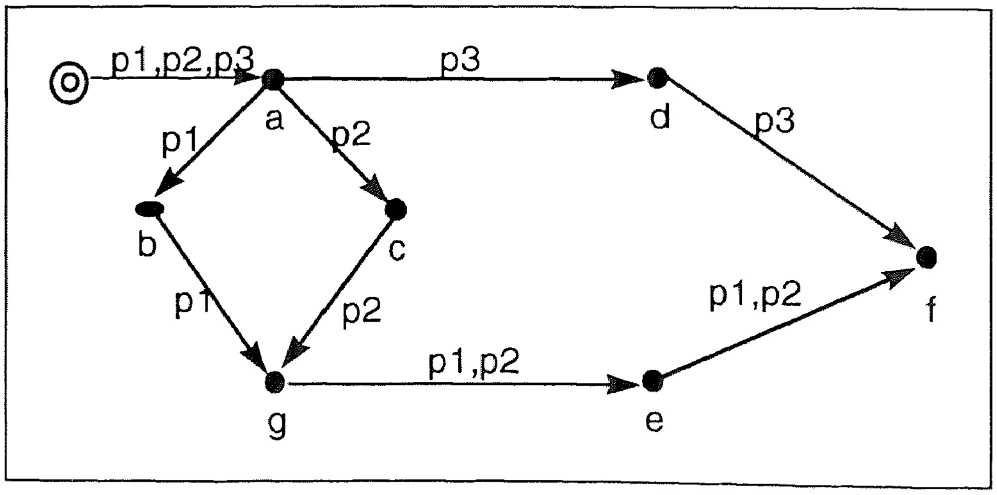


Figure 30 A resource dependency graph.

6.4.2 Resource Assignment Using Maximum Bipartite Matching

If considering resources as one set and the tasks as another set, then assigning resources to tasks can be done in terms of maximum bipartite matching.

The maximum bipartite matching is a problem of finding a maximum number of matches between two sets, L and R [Cormen90] (Figure 31). In Figure 31, the set (a) has a matching with cardinality of 2, and the set (b) has a maximum matching with cardinaliy of 3.

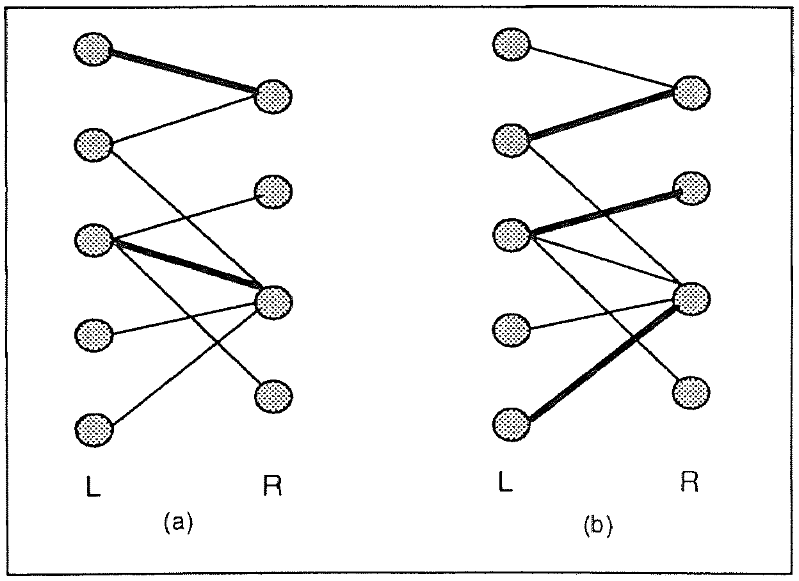


Figure 31 Maximum bipartite matching.

The set of resources and the set of tasks have an n to m relationship. That is, many resources can be assigned to the same task, and one resource can be assigned to many tasks.

The resource assignment using maximum bipartite matching needs to interact with the task ordering by topological sort. If treated separately, the former concentrates assigning resources without considering the task ordering and the latter concentrates on task ordering without considering resource contention. Only coupled with the task duration, the task dependency graph, the set of restrictions, and so on, resources can be assigned to tasks meaningfully.

6.4.3 Determine Process Capacity Using Flow Network

One particular problem in this research is to study the resource capacity usage of the process. Specifically, given a proposed process, what is the resource capacity usage? Given resource restrictions, such as a pre-defined net process time, where should resources be added in order to execute the whole set of tasks within budget and within schedule? What is the effect of adding these resources? Where is the process bottleneck?

One way of calculating resource capacity usage is by finding out the float of the paths as discussed in Chapter 4. An alternative is to consider the resources and resource capacities as a flow network. The resource utilization can then be studied in terms of the maximum flow problem in a flow network.

The maximum flow problem is to find the maximum flow from source(s) to sink(s) of a graph under certain capacity restrictions. For example, we may want to find out what is the greatest rate at which material can be shipped from the source to the sink without violating any capacity constraints. If we think of resources (personnel, facilities, equipment, etc.) as "pipes", and tasks as materials, we can then try to find answer to the following question: What is the greatest rate that tasks can be executed by the personnel within the constraints?

In fact, the capacity issue can be translated to many maximum flow questions as shown below.

- What is the maximum flow of the current network with all the capacity restrictions?
- Where is the bottleneck? Which capacity (between which pair) should be increased and by how much?
- Some of the flows are way under capacity, how to rearrange the flow network so that the capacities are highly utilized?
- What is the effect of adding (or reducing) capacities to the network? Can this increased capacity or capacity usage result in reduced net process time?
- Rearranging the flow network may result in more (or less) capacity being needed; adding capacity to the network may result in the flow network being rearranged. What is the optimal point?

CHAPTER 7

CONCLUSION AND FUTURE RESEARCH

A software process model must possess capabilities in three major categories, namely, representation capabilities, comprehensive analysis capabilities and forecasting capabilities. The software process modeling research, therefore, can be conducted along the three dimensions that provide these capabilities: process representation, process validation and process optimization (Figure 32).

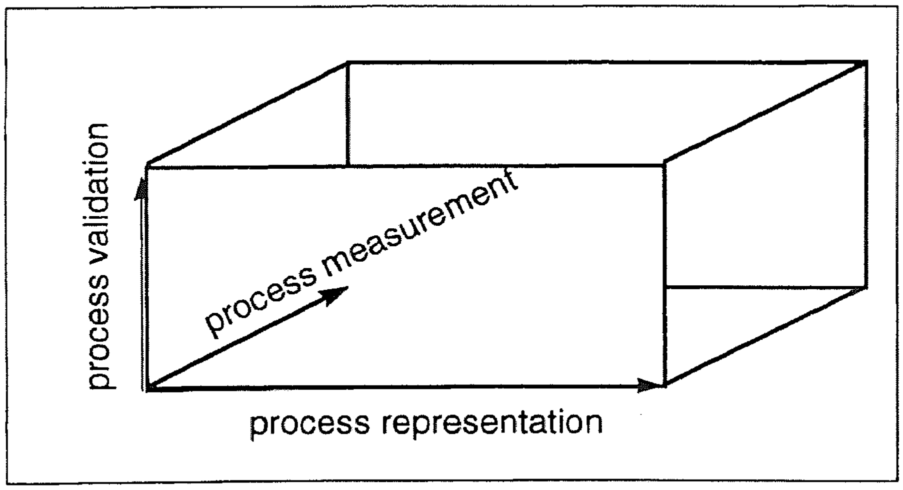


Figure 32 Process modeling dimensions.

Current software process modeling research has made significant progress towards process representation, and to a certain degree, towards process validation. However, few satisfying results have been achieved along the process measurement dimension.

The goal of process modeling should not stop at the process representation level. Describing a process concisely and succinctly is just a first step in studying the process. Process modeling should not stop at the process validation level either. A correctly described process to its completeness does not mean that the process is a good process. The research has to go further. Given a described process, we should be able to answer the following questions: Where is the bottleneck? Where are the slacks? Are the resources being used to their full potential? Are the resource capacities exceeded? Where is the risk area? What are the effects of changing certain parameters? Following answers to these diagnostic questions, an improved process with predictable behavior can be derived. This process improvement is an ongoing activity - after a process is invoked, the process execution is closely monitored and the above questions are re-examined based on the current performance as measured by the collected data. The process is further modified, if necessary.

7.1 Conclusion

A process modeled project management environment enables software development activities to be fully integrated to the process established by an organization or by the whole software development community. Process execution steps form the backbones of a project breakdown structure, based on which the project is decomposed into tasks. Resource assignment to the tasks and scheduling of the tasks are done using scheduling algorithms and techniques. This treatment of tasks is based on a scheduling model in industrial engineering.

From a scheduling model's perspective, software development activities consist of a set of tasks (τ), a set of resources (ρ), and a set of constraints (γ). A process, P, is then a function of all the sets interacting with each other: $P = \{\tau, \rho, \gamma\}$. These sets can be readily "plugged in" to a machine model in a Flexible Flow Shop environment by considering τ as jobs, ρ as machines, and γ as constraints. Study of interactions of these sets is thus done in terms of different flexible flow shop machine models. The outcome of these interactions are reflected through a set of measurement criteria, in particular, the make-span, the float and the bottlenecks. The make-span tells us how long the project is expected to take, the float indicates where the slacks occur, and thus how resource capacities are being utilized, and the bottlenecks point out the area where management focus should be directed. These measurements enable us to quantitatively compare and judge the end results from modification of the process, such as rearrangement of a certain project tasks or resource assignments, whether in its process description stage, process simulation stage or process execution stage.

Description of combining project management and process modeling into an integrated entity in Chapter 2 is followed by a detailed description of deterministic resource models in Chapter 4. The description starts from a simple model with single resource and lots of assumptions and extended to a more general and realistic operating environment with multiple resources, multiple tasks (not necessarily from the same project), and assumptions removed but constraints added. Software development activities often exhibit stochastic behavior. The statistical analysis of these uncertainty is dealt with in Chapter 5.

An integrated process modeling environment is presented in Chapter 6. This sets up the environment where the process can be described, simulated, measured and modified. A feedback look from the actual execution is the key to this process modeling environment. This helps to fulfill the requirements that a process model not only has to be able to predict the process outcome, it also has to react quickly to environmental changes.

Process description using graphs is adopted in this research. While there are many mechanisms to formally describe a process, process description using graphs lends itself to the convenience of studying the process in terms of such established graphical project management schemes as CPM and precedence diagrams. It also allows many established graph theories and algorithms to be used for the process analysis and task manipulations.

The framework established in this research has been experimented with a real world software development process modeling initiative as part of the organization's Process Improvement engagement. Description of this initiative is done in Appendix A as a case study for this research.

7.2 Research Contributions

As discussed in Chapter 2, a good process comprises both a managerial framework and a technical framework. Managerial frameworks are found in many software development organizations; yet technical frameworks are still a topic of interest in the research community. As we move to higher SEI capability levels, good technical frameworks are essential.

An organization's software development capabilities are measured by its level of capability maturity. At SEI level 1 and level 2, no technical frameworks are necessary. At level 3, the *Defined* level, precise and powerful description of processes are needed. Since no manipulation of these processes are required at this level yet, the process description of this level is mainly for understanding of the process. Therefore, a formal process description mechanism is not absolutely needed, but a good process description framework would help the organization move to the next level, the *Managed* level. At this new level, processes need to be formally examined and analyzed, and comprehensive process measurements need to be provided to set the stage for continued improvements in the next level, the *Optimizing* level. Processes at level 5 should have quantitative forecast capability to predict the impact of any process changes.

This is the area where a contribution of this research has been made. Continuing from previous work by Delcambre [Delcambre94] and Mills [Mills96], where process description and analysis were done using a task system template, this research has concentrated on setting a technical framework for process quantitative measurement and comparison, thus helping organizations fulfill the technical requirements of the SEI level 4 and level 5, and in doing so, has advanced research in the area of process forecast and prediction, pushing software process modeling one step further. By integrating process modeling into a software development environment, this research has also bridged the gap between research in the academic world and application in the software development community, thus technically achieving a process centered software development environment.

This research has adopted the following approaches:

1) Integration of project management with process modeling.

Project management and process modeling were treated as two separate subjects in the software research and development community. Researchers in the area of software development process modeling were reluctant in "crossing the border" to tap into the techniques and infrastructure already built into the area of project management. As a result, processes in the software development industry have been mainly used as a policy or guidelines while project management is being used to track development activities.

Because of this separation, the feedback loop as described earlier is being cut, i.e. data from the development activities and environmental changes cannot be put back into the process during the process execution stage. Although project management software has been used to keep track of projects, the computer power has not been fully tapped from the standpoint of process improvement.

In this research project management has been integrated into a process model driven environment. This process centered, integrated project management scheme allows established processes being carried out through project execution tasks, and the results from project execution being put back to the process for process monitoring and modifications.

2) Usage of process modeling techniques from industrial engineering.

Scheduling theory and techniques have been used extensively in industrial engineering. In a manufacturing process modeling environment, jobs are dispatched to different machines according to the capabilities and capacities of the machines and the specified constraints. Different scheduling models have been set up to deal with various types of situations. This modeling technique has been studied and used in software development process modeling in this research. The scheduling models are utilized by considering tasks as jobs, and resources in a software development organization as resources in a manufacturing environment. Different process models, both deterministic and stochastic, are then studied. Measurements criteria have been set up so that the processes obtained by manipulating parameters, resource assignments, tasks arrangements, constraints, and so on, can be quantitatively compared and a judgment can be made based on the semantics.

This inter-disciplinary approach has helped to set a stage for further studies of process modeling using scheduling theory and algorithms.

3) Introduction of an integrated process modeling environment.

In order to simulate processes by manipulating process models, an integrated process modeling environment has been described. This environment integrates tools for all aspects of project management, process engineering and software engineering. Examples of these include tools for process description, process analysis, process simulation, process monitoring, data collections, and many traditional CASE tools used for project management, problem tracking, software development, and software configuration. This environment is essential for a process driven software development. It provides direct feedback from software engineering activities to process engineering activities, or vice versa. It is the base for planning control and risk management. Although a complete implementation of such a system is beyond the scope of this research, the concept proved by this research and the proposal outlined in this dissertation will provide guidance for further study and implementation of an integrated process modeling environment.

7.3 Research Results and Impact

As a problem "from the real world and back to the real world," this research intends to address a software process issue in a division of a telecommunications company. The "magic formula" for the organization's superior product quality and on-time delivery is its adherence to software development processes. As a continued effort in process improvement, *systematic* process control becomes a high priority.

This research has demonstrated that a sound technical process modeling framework not only provides solution to the systematic process control problem, it is also the key to sustain the organization's process capabilities. The result of this research has, therefore, become a proposal to be submitted as part of the organization's Process Improvement (PI) efforts in preparation for the 1998 TCS regional competition. TCS (Total Customer Satisfaction) is the company's initiative which calls for "innovation and smart way of doing business."

The impact of realization of a technical framework would be far beyond the division level. Implementation of the comprehensive process modeling environment would help not only the division but also the rest of the company to achieve and sustain top level SEI maturity, thus to enhance their product development capabilities. The success story can even go beyond the company and make a bigger impact to the whole software development community in its efforts of product quality improvement.

7.4 Direction for Future Research

Extending the scope of the research described in this dissertation, the following identifies future opportunities for its continuation.

The call for an integrated process modeling environment has been made and its validity has been testified in this research. Implementation of such a system, however, requires combined efforts from a team of software developers. Many practical issues, such as budgeting and profitability, need to be worked out before this type of systems becomes a reality.

This research has touched upon the stochastic behavior of software process models. Research in this area needs to be carried further. Software development activities are full of uncertainties. Tackling these volatile process models is crucial in software risk management.

Cost control is an important aspect of any software development endeavor. Constraints related to costs, such as duration sensitivity, lateness charges, benefit of hiring, etc. will directly affect the outcome of a process. Therefore, cost control needs to be considered in future process modeling research.

When describing resource assignments and task sequencing in both the deterministic and stochastic models, this research adopted a set of assumed constraints. A formal method of description for a variety of complex constraints is needed in order to study the impact of the constraints to process models more effectively. This description can be done either mathematically or by way of formal languages.

Determining resource capacity usage using Flow Network has been touched upon in this research. It is an interesting alternative for determining and controlling resource capacity usage and needs to be studied further.

Research is an ongoing activity, with earlier research results serving as step stones and later researches carrying them on by modifying, improving and enhancing earlier research results. It is hoped that this research has served as a step stone for future researches. It is hoped that a "silver bullet" emerges as the entire software research and development community moves forward towards a higher capability maturity level.

APPENDIX A

EMPIRICAL STUDY OF A SOFTWARE PROCESS MODEL

This appendix introduces the case study originated from a software development division within a major wireless telecommunications corporation. As a case study experiment, a software development group and its project has been chosen.

A.1 Business Practice

The particular software development group under study is engaged in a satellite based personal communications system, which consists of four major components: a space network formed by Low Earth Orbit satellites, a land based network formed by ground stations (also called the Gateways), communications devices such as hand-held telephones, and a data-network to support system operations. The group under study is contracted for the software development of a major portion of the Gateway.

System level requirements have been handed down from the system requirements group, who, through years of studies, has laid out a framework for all the components to work together. Development is then carried out in component level organizations. The components are tested locally and then submitted for system level integration.

The software development for the Gateway is on top of an existing hardware platform, with new software being added and existing software being modified or deleted. The software is released in phases, with each phase of delivery carrying more functionality, or features. The development process follows a water fall model and consists of the following process steps:

1. Software Requirement Specification (SRS)
2. Interface Control Design (ICD)
3. High Level Design (HLD)
4. State Machine Design (SMD)
5. Low Level Design (LLD)
6. Coding (CODE)
7. Process Test Design (PTD)
8. Process Test Execution (PTX)
9. Feature Test Design (FTD)
10. Feature Test Execution (FTX)

Once the High Level Design in step 3 is finished for each feature, the development is carried out further by parallel development teams, each team is responsible for a certain sub-component. All the components have to communicate with each other through the communications protocols and messages designed in the Interface Control Design stage. Steps 4 to 8 are followed by each team, if applicable. Step 9, Feature Test Design, can be done any time after step 3 is finished. The last step, Feature Test Execution, is carried out after all the teams have finished step 8 for that particular feature, provided step 9 is also finished. This process is illustrated in Figure 33.

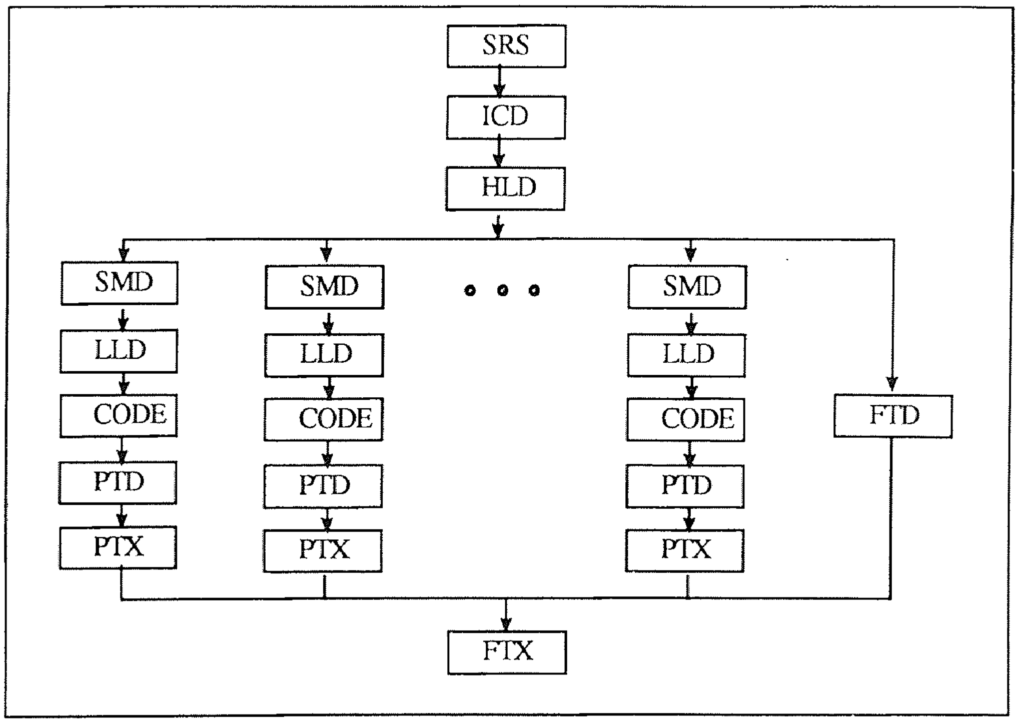


Figure 33 A simplified process example.

Since there are many features being developed in parallel, members in various teams are engaged in different stages of development for different features.

According to the company's quality control policy, all the deliverables, including internal and external documents, coding, test plans, test cases, etc. need to be inspected in an inspection process, called Fagan Inspection. The inspection is normally attended by four or more people. A certain amount of time is given for preparation before the inspection. When the engineer finishes one delivery and is waiting for its inspection, he/she typically goes into the next step and conducts some preliminary work.

After feature test of one or more features, a load carrying all these features is delivered to an internal testing organization for rigorous component level tests.

Note, from the higher level of the development organization, the *product design* and *product test* are two process tasks, with the task, *product design*, followed by the task, *product test.* These two tasks are then decomposed into smaller tasks, and therefore form sub-processes within a process. In this sense, what is described above (see Figure 33) is a sub-process for designing a product; the process being followed by the testing organization is yet another sub-process. Interacting with both the *product design* and *product test* is a sub-process for *problem reporting and resolution.*

A.2 Process Execution Challenge

Software development processes face additional challenges that other processes, such as manufacturing production processes, may not have. In a machine shop environment for example, the process of execution, once established, is fixed for the most part, and the capacity of each machine can be calculated or measured. Risks, such as machine breakdown and material supply shortage, are factored in the production planning. Software development processes, on the other hand, are more subject to changes and abnormal situations.

The software development organization for this case study is no exception. Handling those exceptional situations while carrying on with the normal process execution has become a challenge. A detailed look at the operational environment, or semantics, can make us understand the need for a software process modeling effort.

The following software development challenges were carefully studied:

1) The normal development process can be interrupted by many other events: for example, problem reports can be generated from the field, or from the various testing organizations. If the problem reports need immediate attention, the development process will be put on hold. Setting aside a special team to address those problems seem to resolve

this particular issue, but it may not be practical in terms of operations or budgeting. In reality, certain problems may be best addressed by the developers themselves. From the project delivery and resource capacity usage's point of view, a separate team solution satisfies one measurement (make-span) at the high cost of the other (float time).

To compound the situation, interruption of one task may lead to delays of other tasks, causing a rippling effect.

2) Project task duration is estimated. With enough data collected, the accuracy of estimation can be improved, but it is still estimated, not calculated or measured. Therefore, the actual task execution duration may be different from the estimation, and due to the complex nature of the project, sometimes these differences are quite large - it is either overestimated or underestimated.

3) Other than the development equipment the major resource for software development is the manpower - people. Yet, these resources are subject to a variety of changes. People can come and go; they can take sick leaves; they can be promoted or moved to a different department or company. Their availability or absence are not even completely under their own control. Therefore, the resource assignment has to be updated from time to time, causing rippling effects to the resource dependencies, resource utilization and delivery date.

4) Task assignment is based on people's skill levels. While, say, the production rate of a particular machine is measurable, there is no accurate and convenient way of measuring people's skill levels. Any attempt of such measurements, such as by education levels or by years of experiences, is a rough approximation. Task assignment is also based on people's interest and specialties. While we can assign jobs to machines based on the

machine's functions, we can't assign tasks to people without considering their interest, their training need, the job rotation need, and many other human related issues. Certain tasks can only be performed by certain people, or if changed, the execution duration may need to change also. This not only poses challenges in coming up with a comprehensive set of constraints, it also causes more uncertainty in process execution.

5) During the course of project execution, the task breakdown structure may need to be changed, nullifying many previous dependency relationships and resource assignments.

These software development and project management difficulties can explain the project delays experienced by many organizations. However, these project delays are not tolerated by an SEI level 5 organization, and product quality cannot be compromised either.

In order to deal with these operational issues, a big portion of the manpower in the group under study has to be devoted in project tracking, process control, and process monitoring, causing a major development overhead. Even with the help of a project management tool, for a complicated project this size, the traditional project management method is far from satisfaction. What is needed is a process environment that can systematically adjust to the situation changes, quickly and effectively.

A.3 Process Centered Project Management Setup

As described previously, the first step in combining project management to a process modeling environment is setting up a process execution steps, or *process tasks*.

An example of a simplified process described in Section 0 for the target group under study is set up as shown in Figure 33.

The project work breakdown structure (WBS) is obtained next. This WBS is project specific. However, Each project task should fit in one of the process task above. For example, different modules of the system may need to be modified depending on the project. When the amount of work get large, the modules are grouped (or breaking down) into several project tasks, such as, coding of modules A-J, coding of modules G-I, and so on. All of these are then mapped to the CODE process task for the particular sub-component. Different set of modules may be mapped to a process task in a different sub-component.

Note that the functional dependency of these project tasks may be established automatically by the links from their respective process tasks. See Chapter 3 for detailed descriptions.

The functional dependency set up previously forms a directed acyclic graph (DAG). During the next step, duration for each project task is assigned. This is recorded for each node of the graph. This is followed by a resource assignment step, when initial resource assignment is associated with each task. The purpose of this crude resource assignment is to set up the basis for later graph manipulation and fine-tuning by establishing an initial resource dependency graph.

A.4 Process Execution, Measurement and Analysis

The heart of a process modeling environment is a process modeling engine which incorporates different algorithms for process manipulation. It also has all the measurements calculated for the user to compare the processes and to see the impact to the whole project by changing parameters, such as constraints or resource assignments.

The bulk of this dissertation has been devoted to discussions of measurements. In terms of project management interest, the mangers want to know how long a project can take and how much manpower is needed. In case of environmental changes, such as sudden shortage of resources, they want to know what is the impact to the whole project. From an operational perspective, they want to know whether the teams are over-staffed or understaffed. All these translate into the set of measurement criteria. The major ones are the make-span (the longest duration for the project), the float (the slack time which can be used to calculate the resource capacity usage), the critical paths and bottlenecks.

To deal with uncertainty, measurement of stochastic models are proposed. These measurements give managers a confidence level, or a range of answers when the process is being manipulated. Through a friendly user interface, the tasks manipulation should become very easy. An integrated process modeling environment (see Chapter 6) ensures that data from actual process execution are collected and used on a continual basis.

To completely implement a comprehensive and user friendly process modeling environment requires a team effort. However, once the theoretic framework as discussed in this dissertation is adopted, the system implementation becomes a matter of endorsement from the company or a sponsor for the development effort.

REFERENCES

[Akhras92] F. N. Akhras, et al. *Towards Dynamic Generation of Knowledge-Based Environments for Software Process Assistance*, IEEE, 1992.

[Applegate91] D. Applegate, W. Cook, "A Computational Study of the Job-Shop Scheduling Problem," *ORSA Journal on Computing*, vol. 3, pp.149-156.

[Bandi93] S. C. Bandinelli, A. Fuggetta, C.Ghezzi, *Software Process Model Evolution in the SPADE Environment*, IEEE 1993.

[Boehm86] B. Boehm, "A Spiral Model of Software Development and Enhancement," *ACM SIGSOFT Software Engineering Notes*, August 1986.

[Bohm80] D. Bohm, *Wholeness and the Implicate Order*, Ark Paperbacks, Boston, 1980.

[Brooks87] F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, Vol 20, No. 4, pp.10-19, April 1987.

[Chris92] M. Christiansen, S. Delcambre, E. Demirors, O. Demirors, M Tanik, *Software Development with Transformable Components*, IEEE 1992.

[CaCo93] B. Cain, J. Coplien, *A Role-Based Empirical Process Modeling Environment*, IEEE 1993.

[Cormen90] T. Cormen, C. Leiserson, R. Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[Delcambre94] S. Delcambre, *A Software Process Modeling Framework as a Basis for Process Analysis and Improvement*, Southern Methodist University, 1994.

[Demi92] E. Demirors, O. Demirors, W. Yin, M. Tanik, D. Yun, *An Alternative Software Development Model Supporting Software Evolution*, IEEE 1992.

[Dogru92] A. Dogru, S Delcambre, C. Bayrak, M. Christiansen, M Tanik, *The Development of an Integrated System Design Environment*, IEEE 1992.

[Dsn93] M. Dowson, *Software Process Themes and Issues*, IEEE 1993.

[Dun90] R. Dunn, *Software Quality Concepts and Plans*, Prentic-Hall, Englewood Cliffs, NJ, 1990, pp. 142-145.

[Earl95] A. Earl, A. Christie, "An Approach to Classifying Graphical Process Modeling Notations," *SEI Technical Reports*, 1995, pp.109-123.

[Gibbons91] A. Gibbons, *Algorithmic Graph Theory*, Cambridge University Press, 1991.

[Fagan86] M. E. Fagan, "Advances in Software Inspections," *IEEE Transactions of Software Engineering*, Vol SE-12, No. 7, July 1986.

[Garg94] P. K. Garg, P. Mi, T. Pham, W. Scacchi and G. Thunquest, *The SMART Approach for Software Process Engineering*, IEEE 1994.

[Garg96] P. K. Garg and Mehdi Jazayeri, *Process-Centered Software Engineering Environments*, IEEE Computer Society Press, 1996.

[Hatono92] I. Hatono, et all, "Modeling and On-Line Scheduling of Flexible Manufacturing Systems Using Stochastic Petri Nets," *IEEE Transactions on Software Engineering*, 17(2) 1992, pp.126-131.

[Heim91] D. Heimbigner, "The Process modeling Example Problem and its Solutions," Proc. 1st International Conference on the Software Process, IEEE Computer Society Press, October 1991.

[Hoare85] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International, Englewood Cliffs, NJ, 1985.

[Hodson92] William Hodson, *Maynard's Industrial Engineering Handbook*, McGraw-Hill, 1992.

[Hooc94] S. M. Hooczko, B. J. Hirsh, "Taking Inspections to the Limit," Motorola 7th Software Engineering Symposium, 1995.

[Huff88] K. Huff and V. Lesser, "A Plan-Based Intelligent Assistant That Supports the Process of Programming," ACM SIGSOFT Software Engineering Notes, Vol.13, Nov. 1988, pp.97-106.

[Humphrey88] W. S. Humphrey, "Characterizing the Software Process: A Maturity Framework," *IEEE Software*, 5(2), March 1988, pp.73-79.

[Humphrey90] W. S. Humphrey, *Managing the Software Process*, Addison-Wesley Publishing Company, 1990.

[Jacc93] M. L. Jaccheri, R. Conradi, *Techniques for Process Model Evolution in EPOS*, IEEE 1993.

[Kaiser87] G. E. Kaiser, P. H. Feiler, *An architecture for Intelligent Assistance in Software Development*, Proceedings of 9th ICSE, Monterey, CA, April 1987.

[Kaiser88] G. E. Kaiser, et al. *Database Support for Knowledge-Based Engineering Environments*, IEEE Expert, 1988.

[Kaiser88a] G. E. Kaiser, "Rule-Based Modeling of the Software Development Process," *Proceedings of the 4th Int. Software Process Workshop*, New York, NY, 1988, pp.84-86.

[Kellner89] M. Kellner, "Software Process Modeling: Value and Experience," *SEI Technical Review*, 89(23), 1989, pp.23-54.

[Kellner93] Marc Kellner, *Tutorial: Software Process Modeling*, Software Engineering Institute, Carnegie Mellon University, 1993.

[Kramer90] B. Kramer, Luqi, *Petri Net-Based Models of Software Engineering Processes*, IEEE 1990.

[Lehman87] M. Lehman, "Process Models, Process Programs, Programming Support," *Proceedings of 9th International Conference on Software Engineering*, 1987, pp.14-16.

[Lin89] Chi Y. Lin, et al. *Computer-Aided Software Development Process Design*, IEEE 1989.

[Luqi92] Luqi, *The Management of Uncertainty in Software Development*, IEEE 1992.

[Lonch90] J. Lonchamp, K. Benali, C. Godart, J. C. Derniame, *Modeling and Enacting Software Processes: an Analysis*, IEEE 1990.

[Madh90] N. H. Madhavji, et al., "Prism = Methodology + Process-oriented Environment," McGill University, Montreal, *Proceedings of the 12th ICSE*, Nice, March 1990.

[Matsu90] Y. Matsumoto, "Toshiba Fuchu Software Factory," *Modern Software Engineering*, Van Nostrand Reinhold, New York, 1990, pp.479-501.

[Mills96] Stephen Mills, *Resource-Focused Engineering of Reliable and Efficient Process Systems*, Southern Methodist University, 1996.

[Milner80] R. Milner, "A Calculus of Communicating Systems," *Lecture Notes in Computer Science*, Vol. 92, Springer-Verlag, New York, 1980.

[Moder83] J. J. Moder, et al., *Project Management with CPM, PERT and Precedence Diagramming*, 3rd Edition, Van Nostrand Reinhold Company, 1998.

[Mou96] Gary Mou, "A Graph Based Process Representation for Process Modeling," *Proceedings of the Second World Conference on Integrated Design and Process Technology*, Society for Design and Process Science, 1996, pp.370-375.

[Ng90] Peter A. Ng, Raymond T. Yeh, *Modern Software Engineering, Foundations and Current Perspectives*, Van Nostrand Reinhold, New York, 1990.

[Ost87] L. Osterweil, "Software Processes are Software Too," *Proceedings of 9th International Conference on Software Engineering*, IEEE Computer Society Press, 1987, pp.2-13.

[Ould88] M. Ould, C. Roberts, *Defining formal models of the software development process, Software Engineering Environments*, Brereton P. (ed.), Ellis Horwood, 1988.

[Pinedo95] Michael Pinedo, *Scheduling - Theory, Algorithms, and Systems*, Prentice-Hall, 1995.

[Petri87] C. A. Petri, "Concurrency Theory," *Lecture Notes in Computer Science*, Springer-Verlag, 1987.

[PMI96] PMI Standards Committee, *A Guide to the Project Management Body of Knowledge*, Project Management Institute, 1996.

[Royce70] W. Royce, *Managing the Development of Large Software Systems*, IEEE WESCON, Aug. 1970.

[Sakurai85] J. J. Sakurai, *Modern Quantum Mechanics*, Benjamin / Cummings, 1985.

[Shih95] C. Shih, "Fuzzy Multiobjective Optimization Techniques and Process for Engineering Design," *Proceedings of the First World Conference on Integrated Design and Process Technology*, Austin, 1995, pp.57-62.

[Soma95] H. Soma, et all, *Schedule Optimization Using Fuzzy Inference*, IEEE, July, 1995.

[Tajima90] D. Tajima, Y. Usuda, F. Tsunoda, S. Ebina, "Hitachi's Software Factory Tools for COBOL: An Introduction to Skips/SDE for Business Applications," *Modern Software Engineering*, Van Nostrand Reinhold, New York, 1990, pp.421-447.

[Tanik87] Murat Tanik, "In Search of Silver Bullet," *IEEE/ACM Fall Joint Computer Conference Proceedings*, Dallas, TX, October 1987, pp.686-687.

[Tanik91] Murat Tanik, E. Chan, *Fundamentals of Computing for Software Engineers*, Van Nostrand Reinhold, New York, NY, 1991.

[Tanik96] Murat Tanik, Raymond Yeh, Franz Kurfess, et. al., "Issues and Architecture for Electronic Enterprise Engineering (EEE)," *Proceedings of the Second World Conference on Integrated Design and Process Technology*, Society for Design and Process Science, 1996, pp.57-62.

[Yeh90] Raymond Yeh, Peter Ng, *Modern Software Engineering*, Van Nostrand Reinhold, New York, NY, 1990.

[Yeh94] Raymond Yeh, Suzanne Delcambre, Murat Tanik, "Cosmos: An Architecture for Improving Process Capability Maturity," *3rd International Conference on Systems Integration*, 1994.

[Yu94] Eric S. K. Yu, J. Mylopoulos, *Understanding "Why" in Software Process Modeling, Analysis, and Design*, IEEE 1994.

[Zeigler76] B. P. Zeigler, *Theory of Modeling and Simulation*, Wiley-Interscience, New York, 1976.