Technical and managerial complexity increasingly overwhelm project managers. To rein in that complexity, the authors propose PM-Net, a model that captures the concurrent, iterative, and evolutionary nature of software development. It adopts the basic concepts of Petri nets—graphical models of information flow—with extensions to represent both decisions and artifacts.

A Net Practice for Software Project Management

Carl K. Chang, University of Illinois at Chicago
Mark Christensen

Since the early 1990s, the software industry has found it imperative to develop complex and high-quality software in a highly productive and cost-efficient way.[1] The fundamental challenge entails coordinating and controlling available resources to develop satisfactory software within time constraints and at minimal cost.

If this process is undertaken manually, project managers rarely can hope to develop optimal schedules within reasonable time frames. Yet with the growing costs and tightening time requirements of software development, such an effort is necessary: Developing a program with 100,000 lines of code can easily consume more than a year and $5 million. Given such figures, anything that reduces the time and cost by even 5 percent is worth doing.

There is growing concern in the software industry about the lack of an adequate formal model for managing such development.[1–3] According to Capers Jones, most work in software engineering has focused on building computer-aided software engineering tools to facilitate design, implementation, and testing, while "formal methods and tools used by management as the basis for sizing, planning, estimating, and tracking major software projects are often close to nonexistent."[4]

0740-7459/99/$10.00 © 1999 IEEE

Thus, we propose a novel technique—Project Management Net—to generate near-optimal resource allocation and schedules.[5] PM-Net's analysis of project status and decision making are based on genetic algorithms, which are used to emulate genetic evolution mechanisms.

Software Project Management

For the software project manager, the reliable production of large software systems remains a thicket of problems, including

- assembling, training, and motivating a large workforce;
- developing or adopting engineering and management processes;
- developing and maintaining requirements;
- planning, budgeting, and scheduling the project;
- identifying and resolving resource conflicts;
- monitoring the entire project continuously, applying corrective action as new conflicts arise.

In addition, software project management involves the difficulties arising from products and underlying tools of an evolutionary nature. These factors are especially pervasive in large projects.

Project management is a problem-solving activity that involves four steps:

1. Determining what tasks must be done.
2. Deciding how to do them, including in what sequence and by whom.
3. Controlling how they are to be done.
4. Evaluating (or measuring) what was done.

Determining what must be done typically takes the form of a plan, but popular planning strategies fail to address one or more pivotal concerns in software development. A *work breakdown structure*, or WBS, provides a hierarchical view for the whole project but does not identify the precedence of relationships among the work packages.[6] Even the classic, network-based planning models used today—such as PERT (Program Evaluation and Review Technique) and CPM (Critical-Path Method)—are often inadequate for large projects; they are weak in modeling and analysis of the concurrent, iterative, and evolutionary characteristics of a software project.[2,7] The more recent DesignNet, proposed as a formal method to describe the behavior of software development, also falls short by failing to include resource allocation and temporal characteristics.[2]

Deciding how to perform project tasks involves allocating resources and generating schedules. These activities have an intrinsic complexity that stems from scaling problems associated with large software projects. As Murray Cantor explains, "You set a project budget by assigning cost to each WBS item" and then "you assign developers to the WBS tasks."[8]

Parametric tools typically assist in estimating the project budget and macroschedule, while managers tap their experience with the software development process and the personnel involved to assign the staff to tasks. This level of individual experience with developers is often not available for larger programs; without it, assigning developers to tasks becomes incredibly complex even if the data is available.

Decision-making processes become extremely difficult for large software projects that involve hundreds to thousands of tasks, many developers, and a variety of hardware and software resources. When the project size is small, we certainly can generate *feasible* schedules—those that meet all project conditions and constraints—with conventional project planning tools. Yet even with small projects, what is feasible often changes as the project evolves, and what is feasible is often not the optimal assignment of resources, even under static conditions.

Software Management Tools and PM-Net

Over 120 project management tools are now commercially available, offering different functionality and using various platforms.[9] In general, these tools include such functions as project scheduling, resource management, project tracking, and project reporting.

Each tool fits somewhere in this general process:

1. Define the project's WBS.
2. Analyze task precedence.
3. Assign starting date and duration for each activity—schedule.
4. Identify and define resources.
5. Allocate resources manually.
6. Resolve resource conflicts.
7. Obtain approval of the project plan.
8. Establish project baselines.
9. Measure and record progress.
10. Make adjustments to the project plan.
11. Report project information.

Based on traditional project management techniques such as Gantt charts, CPM, and PERT charts, these tools offer excellent recording and reporting

capabilities—but they do not offer higher-order functions. For example, the CPM method fails to work if we impose various resource restrictions on the project network. In addition, the representation of artifacts such as documentation and code is implicit in these schemes. That is, completion of a series of activities produces an artifact, but the artifact is not explicitly represented.

The current tools also treat project scheduling and resource allocation as two separate problems, although they are highly interdependent. This results in an iterative process of resource assignment, schedule evaluation, and reassignment. This task is done manually, making it tedious and error prone.

PM-Net addresses these problems. To capture the concurrent and evolutionary nature of software development, PM-Net adopts the basic concepts of *Petri nets*—graphical models of information flow—with extensions to represent both decisions and artifacts.[10] PM-Net's design accommodates translation of schedules into task precedence graphs, an internal model commonly used for project management tools. Thus, PM-Net can be connected to existing tools through translation and programming on the tools' application program interface.

Our model's expressive capabilities make it a richer, more natural method of communication between software managers and developers. Developers often dismiss schedules built using program scheduling models on the grounds that they are simplistic and rigid. PM-Net allows the representation of branching and iteration precisely for this reason, as well as to allow alternative views of a project's evolution. Another advantage of PM-Net lies in the foundation it gives software developers for building tools that will support and enhance the software process.

PM-Net calculates an activity's execution time and costs according to its complexity constraints.

PM-Net delivers the project management features commonly offered by existing tools. However, with PM-Net, users can also use a set of advanced software management features, such as automation of resource allocation and scheduling based on genetic algorithms, generation of a structured activity network, and prediction, within some range, of the project's future status in terms of cost and schedule for project completion.

Despite industry doubt that a single model can capture all facets of software development,[1,11] it is clear that such a model should do the following:

- Support a variety of management functionality, such as project planning, project scheduling, resource allocation, project tracking, project reporting, and project predicting.
- Reflect that software development is a design-intensive activity. The design process itself is evolutionary in nature in that task definitions and task assignments periodically get revised.
- Describe the parallel and concurrent nature inherent in software development.
- Support abstraction by hiding unnecessary details, thus providing a clear, high-level view.
- Describe a variety of activities and artifacts arising at various phases of software development.
- Be executable.

PM-Net is meant to meet these needs.

PM-Net Structure

To capture the concurrency of the software development process, PM-Net borrows some concepts from Petri nets.[10] However, in PM-Net both the information carried by the tokens as they pass along the network to the nodes and the rules followed for triggering the execution of the nodes as the tokens arrive are different from those of Petri nets.

PM-Net consists of sets of places, constraints, transitions, and arcs.

Places

There are four different place types: abstract activity, atomic activity, product, and decision. An *abstract activity* can consist of a collection of subordinate atomic or other abstract activities. An *atomic activity* cannot be further decomposed.

By grouping related, lower-level atomic or abstract activities together, an abstract activity provides software managers with a modular view, hiding underlying details in large projects. We often use the generic term *activity* without differentiating between abstract and atomic levels.

The third type of place, the *product* place, represents the artifacts created during the software development process. Finally, the *decision* place represents a *success* or *fail* decision after finishing an activity and is used to reflect the iterative nature of software development.

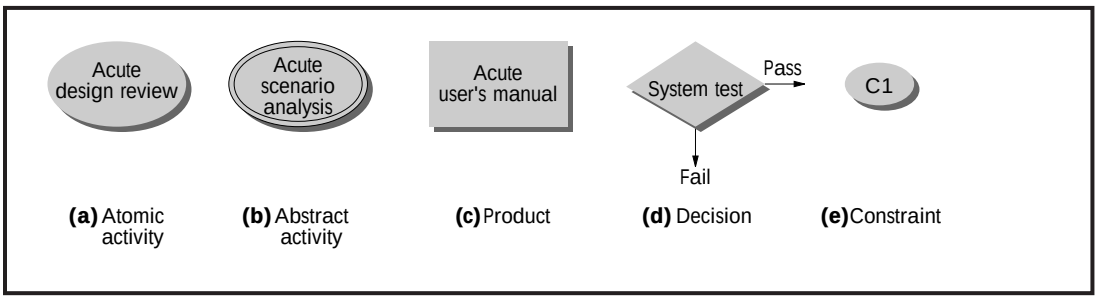

Figure 1. PM-Net graphical symbols of the Acute accounting system.

Constraints

Associated with each activity is a set of constraints that specify the requirements for completing the activity. An abstract activity accumulates constraints from its lower-level activities. The constraints can be further classified as

- *resource constraints*, which specify what kinds of resources are required, and
- *complexity constraints*, which describe how much effort is needed for the activity.

Other types of constraints, such as temporal and precedence constraints, are already embedded in the task precedence graph.

The manager or scheduler must enter the constraints for each activity. Once resources have been assigned in a manner consistent with the resource constraints, PM-Net can calculate the activity's execution time and costs according to the complexity constraints on that activity. Given an optimization goal, usually taken to be some mix of cost and schedule, we use genetic algorithms to find an optimal or nearly optimal project plan. Finally, using PM-Net, the software managers can pre-execute the plan to visualize project progression in advance.

Transitions

The dependency among activities is linked by transitions, product places, and decision places. There are three types of transitions:

- T_I, representing the input transition of an activity.
- T_O, representing the output transition of an activity.
- T_{Do}, representing the output transition of a decision place.

Each transition type has a different meaning and different firing rules.

Arcs

Similarly, arcs for connecting places and transitions can be classified into seven different types.[5] According to Petri net syntax, arcs can only connect a place to a transition or a transition to a place; they cannot connect two places or two transitions.

Graphical Symbols

Figure 1 shows the graphical symbols used in PM-Net as they relate to a hypothetical GUI-based accounting system called Acute. Figure 1a shows the activity of reviewing the design specification—an atomic activity (represented by a large oval) and therefore not comprising subordinate activities. Figure 1b shows the abstract activity (represented by two concentric ovals) of conducting the scenario analysis. We can decompose this abstract activity into at least six lower-level activities:[12]

- Scenario analysis, including scenario elicitation
- Scenario formalization
- Scenario verification
- Scenario generation
- Prototype generation
- Scenario validation

Figure 1c shows a product (represented by a rectangle) of the project, the user's manual. Figure 1d shows a typical pass–fail decision construct (represented by a diamond and two arrows), which would typically be used to perform iterative development. Figure 1e shows a budget constraint (represented by a smaller, elongated oval) that allocates a maximum of $120,000 to the salary of the accounting system specialists.

Figure 2 shows the Acute accounting system, with distinct atomic activities for accounting and for the GUI. Figure 3 combines the two design activities of Figure 2 into one abstract activity.

Notice in Figure 2 five atomic activities—accounting system design and implementation, GUI design and implementation, accounting system testing, GUI testing, and integration testing—and five activities applied to them:

C1: Requires staff with accounting and system design experience. Estimated effort: 10 staff months.

C2: Requires staff familiar with GUI design. Estimated effort: 8 staff months.

C3: Requires testing staff with accounting background. Estimated effort: 3 staff months.

C4: Requires testing staff with GUI design background. Estimated effort: 2 staff months.

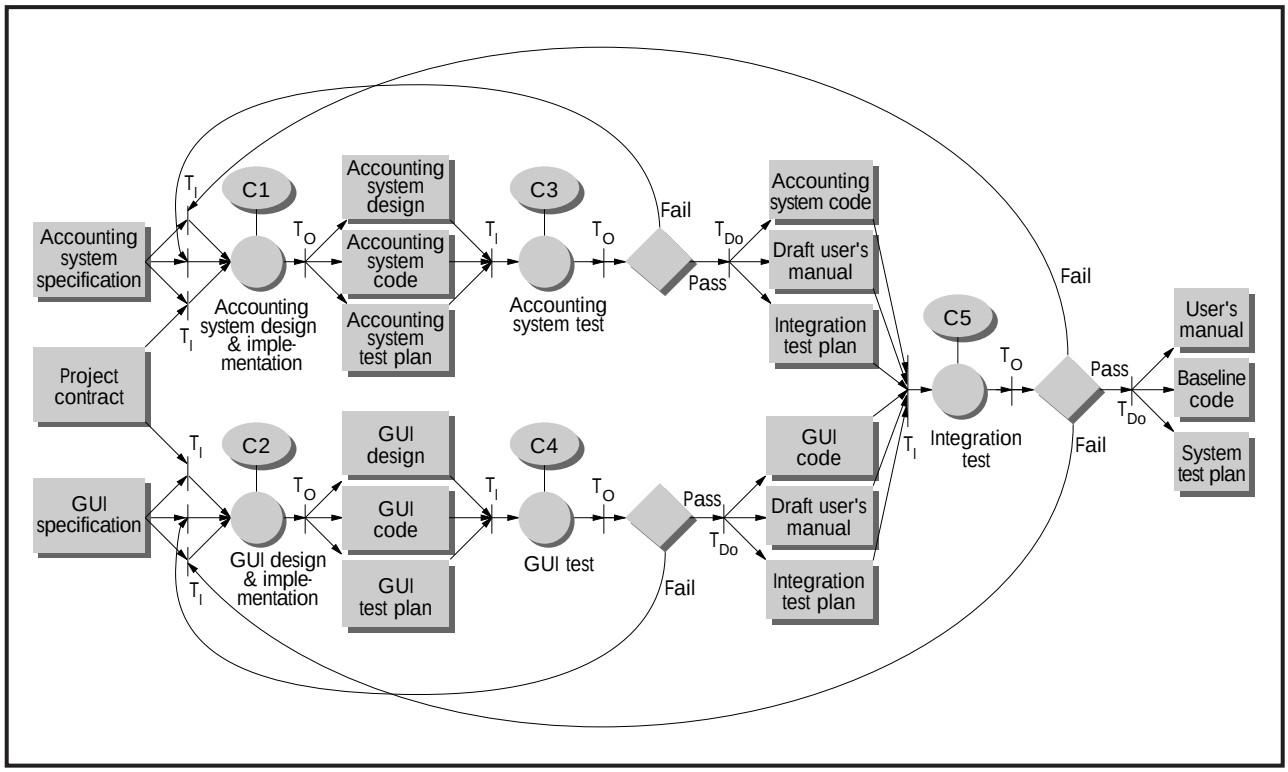

Figure 2. The hypothetical Acute accounting system, with distinct activities for accounting and for the GUI.

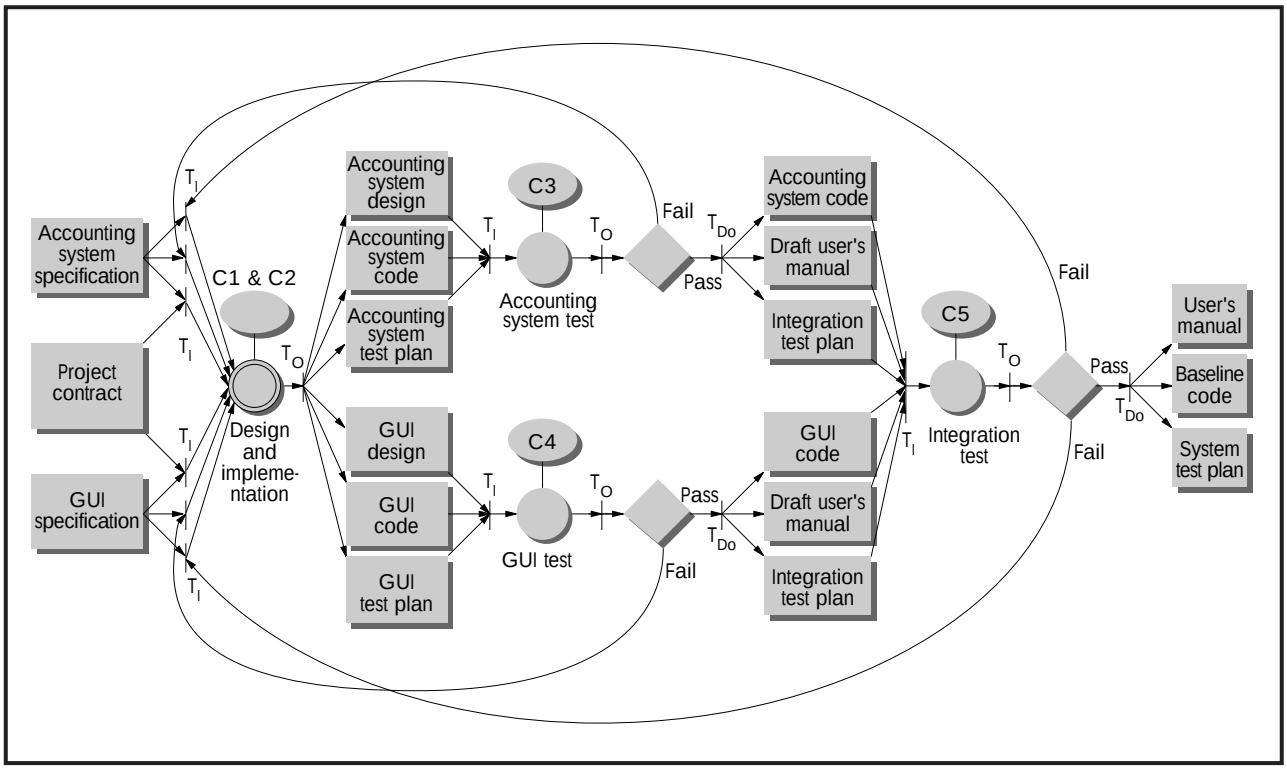

Figure 3. The Acute accounting system combining the two design activities into one abstract activity.

C5: Requires testing staff with integration testing experience. Estimated effort: 2 staff months.

These constraints state the type of individual required as well as the amount of effort required. Data available to the algorithm will determine the depth of specification of resources.

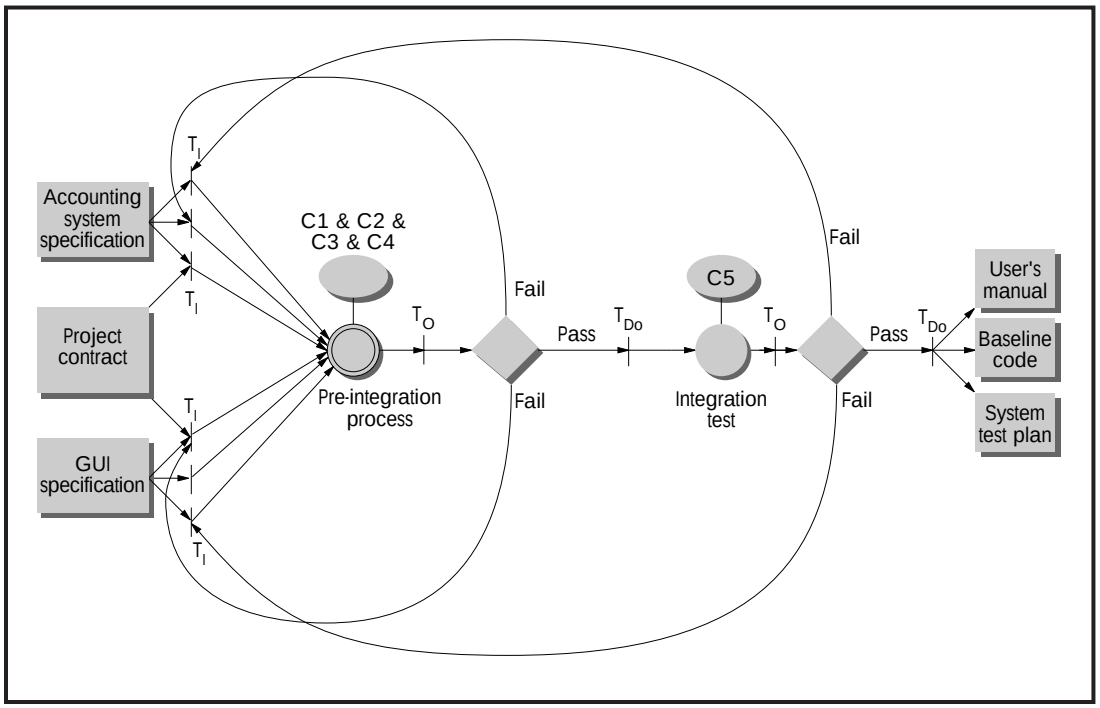

Figure 4. Further reduction of the Acute PM-Net.

Figure 4 shows a further reduction of Figure 3 to a PM-Net in which the activities from design through subsystem test are combined into an abstract activity. Notice that the design implementation/subsystem test—the preintegration process—now contains products externally visible in Figure 3.

Genetic Algorithms

PM-Net uses genetic algorithms to help determine the optimum allocation of project resources. Genetic algorithms have been shown to be a robust solution-space search technique in a variety of optimization problems.[13]

Genetic algorithms emulate these evolution mechanisms:

- *natural selection*—only the fittest survive;
- *reproduction*—established traits are regrouped probabilistically into new combinations by the mechanism of crossover; and
- *random mutation*—allowing for changing environmental conditions and preventing overspecialization.

Genetic algorithms mimic a population of individuals, each representing a feasible solution to the scheduling problem. Each individual is assigned a fitness score, according to how well the individual solves the problem, including the value of the goal function obtained.

Individuals with higher fitness scores have better opportunities to reproduce with others, creating offspring who will inherit some features from their parents. The least-fit individuals are less likely to reproduce and will gradually disappear in successive generations unless mutation produces a positive adaptation. As a result, over many generations, good characteristics are spread throughout the population. By selecting the individuals with higher fitness scores, the most promising areas of the search space of the solution population—in the scheduling example, the most promising schedule options—are explored, while mutation prevents the search from becoming too narrow. In this way, finding solutions to a scheduling problem mimics natural evolution.

Resource allocation and scheduling

Through PM-Net, we can apply genetic algorithms to the problems of project scheduling and resource allocation, as follows:

1. We annotate the task precedence graph with the resource requirements of each task.

2. We calculate the resource allocation and resulting schedules based on this TPG. The allocation and scheduling make use of an employee database and other resources to initially assign random, feasible task allocations.

3. This initial population of schedules evolves through the execution of genetic algorithms until an optimal or nearly optimal match of resources to tasks is obtained.

Specifically, to solve the resource allocation problem, a PM-Net is mapped to a directed, acyclic TPG, as shown in Figure 5. A TPG is represented as an or-

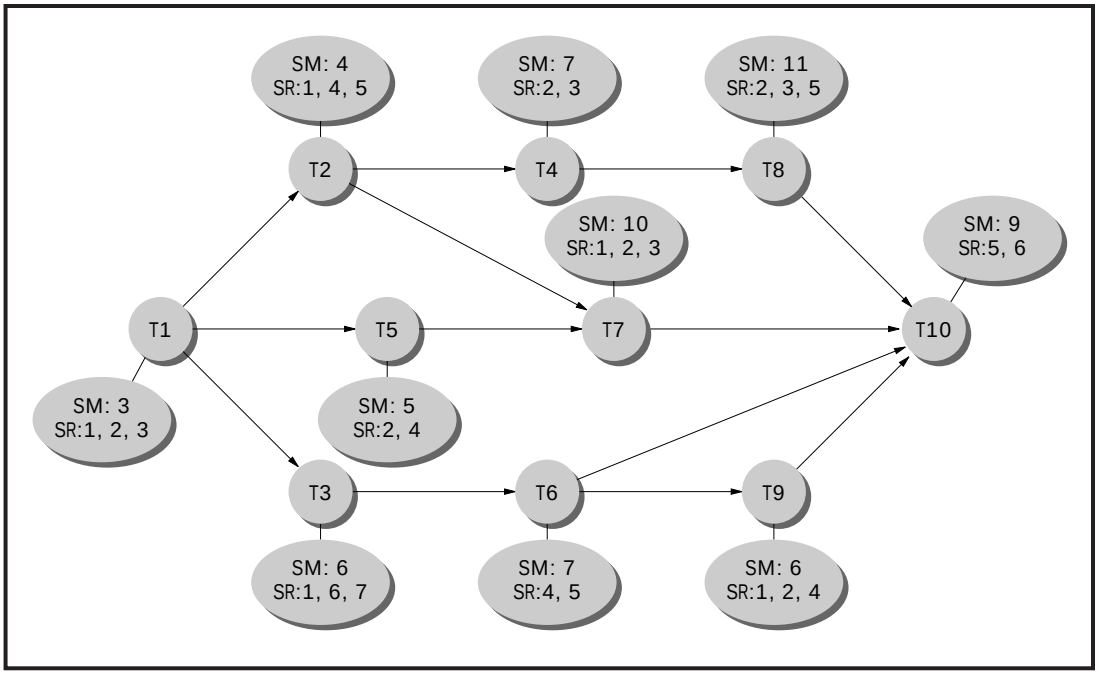

Figure 5. A sample task precedence graph, where SM stands for staff-month and SR stands for skill required.

dered pair (V, E) consisting of a finite, nonempty set of vertices (V) and a finite set of directed edges (E) connecting the vertices. The collection of vertices $V = \{T_1, T_2, \ldots, T_m\}$ represents the set of software tasks to be completed, and each vertex consists of estimated effort (staff-months or working days), required skills, and proficiency of the skill.

Thus, for every $v \in V$, $v = \{v_E, v_S\}$, where v_E represents the estimated effort (staff-months or working days) and v_S represents a list of skills along with the proficiency for each skill and maximal allowable units. The directed edge set $E = \{e_{ij}\}$ (where e_{ij} represents a directed edge from vertex T_i to T_j) implies that a partial ordering or precedence relation (denoted by $\rightarrow$) exists between tasks. That is, if $T_i \rightarrow T_j$, then T_i must be completed before T_j can be initiated.

We have used domain-specific knowledge to improve the convergence rate of the genetic algorithms for resource allocation. This is facilitated by the use of the project schedule itself instead of an encoded bit-string such as a chromosome (also referred to as a *string* or a *solution*). This choice makes the introduction of domain-specific knowledge easier, thereby enabling more efficient exploration of the search space around good points.[14] This knowledge is incorporated into the schedule generation, selection, crossover, and mutation operations to ensure that they will

- always yield legal schedules,
- generate diverse and adaptive schedules, and
- guide genetic operators more directly toward better schedules.

In our approach, each string represents a schedule consisting of two complementary sets of lists: a set of *task assignment lists* and a set of *employee assignment lists*. Each task assignment list shows the group of employees assigned to a particular task in a schedule. Each employee assignment list shows the group of tasks assigned to a particular employee in a schedule.

In our case, the crossover operator works on the employee assignment lists, while the mutation operator works on task assignment lists. Thus, for the sake of efficiency, we keep both sets at the same time, updating the complementary list whenever the other changes.

By avoiding introducing unfit or poor schedules into the populations, this approach confines the search to feasible and promising regions while not overly restricting the search space. Consequently, efficient performance of the search can be achieved along with nearly optimal results. Caution must be applied in introducing domain knowledge, as such knowledge necessarily originates from finite experi-

ence; overuse could reduce the breadth of the search.

The algorithm basically uses small, heuristically initialized populations, proportional reproduction (known as selection), one-point crossover, and single-assignment mutation schemes. In our problem, a *legal* schedule must satisfy the following requirements:

- the precedence relations among tasks,
- conditions and constraints imposed on each task, and
- completeness of the schedule (that is, all tasks must appear in the schedule).

The highest-level genetic algorithm appears in Figure 6.

The lower-level algorithms needed to perform GA-Scheduling are Generate-Initial-Population, Fitness-Evaluation, Crossover, Mutation, Selection, and Refine-Schedule, all of which are described elsewhere.[5]

Algorithm GA-Scheduling

Input: a task precedence graph, TPG = (V, E); an employee database, D_{emp}; and a skill database, D_{skill}. The parameters describing the search are

Pop—population of schedules at the start of the iteration
Newpop—new population
Npop—size of population
$Prob_C$—crossover probability
$Prob_M$—mutation probability
$S_{refined}$—refine the best-so-far schedule by exploring neighborhood

Output: one near-optimal schedule, $S_{near\text{-}opt}$

Begin
1. Generate initial population Pop containing *Npop* schedules.
2. Compute the fitness value of each schedule in Pop.
3. Perform Selection to obtain Newpop.
4. Perform Crossover *Npop*/2 times by randomly selecting two schedules from Newpop and do the crossover with a probability $Prob_C$.
5. Perform Mutation *Npop*/2 times for each schedule with an adaptive probability $Prob_M$ and return the schedule back to Pop.
6. Refine the best-so-far schedule, $S_{refined}$, in Pop.
7. If convergence criteria not satisfied, loop to 2.
8. Output the final near-optimal schedule, $S_{near\text{-}opt} \leftarrow S_{refined}$.
End

Figure 6. The highest-level genetic algorithm, called GA-Scheduling.

Experimental results

To obtain a baseline for purposes of comparison, we also implemented an optimal-scheduling algorithm based on an exhaustive search. This algorithm, which requires exponential time to execute, generates all feasible combinations of schedules and determines which combination produces the best fitness value. The programs were implemented using C++ on a Sun–Sparc 10 workstation.

Table 1 summarizes the experimental results from both the exhaustive-search and genetic algorithms. For the first eight projects, the results from the GA-scheduling algorithm found the optimal solution. In the last three cases, we cannot verify that the GA-derived schedules are optimal because the exhaustive method could not identify the optimal solution; the 11th project would require around 10^{11} years of computation time on a Sparc 10.

For many engineering tasks, such as software project management, near-optimal solutions ob-

Table 1
EXPERIMENTAL RESULTS FROM EXHAUSTIVE SEARCH AND GA-SCHEDULING ALGORITHMS

Project	Number of tasks	Number of programmers	Combinations	Optimal time (min: sec) (exhaustive search)	Optimal time (sec) (GA algorithm)
1	18	9	1.07×10^5	1:01	3
2	18	9	2.92×10^6	28:48	7
3	11	10	3.75×10^6	25:08	11
4	18	9	6.81×10^6	65:58	8
5	30	9	4.73×10^7	674:40	12
6	50	9	1.00×10^8	2394:00	30
7	10	19	2.11×10^8	711:55	7
8	18	10	6.15×10^8	645:32	9
9	15	19	7.60×10^{12}	n/a	18
10	20	9	2.20×10^{12}	n/a	18
11	18	19	6.81×10^{21}	n/a	14

tained from limited computation will suffice. Extensive simulation results by tuning parameters such as population size, mutation, and crossover rates can be found elsewhere.[5]

The combination of the PM-Net representation and genetic algorithms will allow project managers to realize optimal or nearly optimal schedules, without manually exploring the exponentially large search space of all feasible resource-to-activity assignments.

As an extension, it is possible to assign a probability-of-success rate to each decision place in PM-Net. Moreover, PM-Net supports project planning by predicting the future state of the project. We further divide the software project prediction problem into two classes, wherein one or more decision places in PM-Net drive the goal function beyond the domain of acceptable performance. These concepts extend the conventional critical-path notion.

We plan to enhance PM-Net in several ways.

♦ Incorporating additional factors such as variable cost factors, risk management, software quality, and reliability into the framework by extending the resource database, generalizing the goal function, and adding new heuristics. Level loading of resources is an example of such factors.

♦ Developing external linkages that will, for example, allow the status of artifacts to be obtained automatically from the project configuration management system.

♦ Developing the PM-Net tools environment with expanded visualization techniques.

Because such extensions are labor intensive by themselves, we invite software vendors to work with us on adding these capabilities to existing management tool suites.

We believe PM-Net provides a solid foundation from which to attack many issues in software project management. Furthermore, we feel PM-Net serves as a concrete model for building intelligent tools to support and enhance the project management process. ❖

References

1. V.R. Basili and J.D. Musa, "The Future Engineering of Software: A Management Perspective," *Computer*, Vol. 24, No. 9, Sept. 1991, pp. 90–96.
2. L.C. Liu and E. Horowitz, "A Formal Model for Software Project Management," *IEEE Trans. Software Eng.*, Vol. 15, No. 10, Oct. 1989, pp. 1280–1293.
3. D.J. Reifer, *Software Management*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1993.
4. C. Jones, "Software Management: The Weakest Link in the Software Engineering," *Computer*, Vol. 27, No. 5, May 1994, pp. 10–11.
5. C. Chao, *SPM-Net: A New Methodology for Software Management,* doctoral dissertation, Univ. of Illinois at Chicago, Dept. of Electrical Eng. and Computer Science, 1994.
6. R.C. Tausworthe, "The Work Breakdown Structure in Software Project Management," *J. Systems and Software*, Vol. 1, No. 3, 1980, pp. 181–186.
7. B.I. Blum, *Software Engineering: A Holistic View,* Oxford Univ. Press, New York, 1992.
8. M.R. Cantor, *Object-Oriented Project Management with UML*, John Wiley & Sons, New York, 1998, p. 160.
9. R.K. Wysocki, R. Beck Jr., and D.B. Crane, *Effective Project Management,* John Wiley & Sons, New York, 1995, p. 280.
10. J.L. Peterson, "Petri Nets," *ACM Computing Surveys*, Vol. 9, No. 3, Sept. 1977, pp. 223–252.
11. B.W. Boehm and R. Ross, "Theory-W Software Project Management: Principles and Examples," *IEEE Trans. Software Eng.*, Vol. 15, No. 7, July 1989, pp. 902–916.
12. P. Hsia et al., "Formal Approach to Scenario Analysis," *IEEE Software*, Vol. 11, No. 3, Mar. 1994, pp. 33–41.
13. M. Srinivas and L.M. Patnaik, "Genetic Algorithms: A Survey," *Computer*, Vol. 27, No. 6, June 1994, pp. 17–26.
14. J.Y. Suh and D. Van Gucht, "Incorporating Heuristic Information into Genetic Search," *Proc. 2nd Int'l Conf. Genetic Algorithms*, Erlbaum, Mahwah, N.J., 1987, pp. 100–107.

Acknowledgments

We would like to acknowledge the original contributions of Chikuang Chao and the programming effort by Su-Yin Hsieh.

About the Authors

Carl Chang directs the International Center for Software Engineering at the University of Illinois at Chicago. He previously worked for GTE Automatic Electric and AT&T Bell Labs. He is vice president for press activities of the IEEE Computer Society and chairs the Curricula 2001 joint task force of the Computer Society and ACM that is aimed at renovating computing curricula for the next millenium. He was *IEEE Software*'s editor-in-chief from 1991 to 1994.

Chang received his PhD in computer science from Northwestern University. He is an IEEE senior member and a member of the ACM, Upsilon Pi Epsilon, and Sigma Xi. Contact Chang at cchang@computer.org.

Mark Christensen works as an independent contractor in systems and software engineering. He enjoys solving hard problems at the boundaries of hardware and software and has worked on and delivered computer-controlled radio frequency and opto-mechanical systems. Previously, Christensen was head of the software engineering department, including the IT group, and then vice president of engineering at Northrop Grumman's Electronic Systems site.

Christensen received his PhD in math from Wayne State University and his master's in physics from Purdue University. He is a member of the IEEE Computer Society, INCOSE, and SIAM. Contact Christensen at markchri@concentric.net.

IEEE **SOFTWARE**

Call for Articles & Reviewers
Malicious Information Technology: The Software vs. The People
Publication: Sept./Oct. 2000

Software was intended to improve the quality of human life by doing tasks more quickly, reliably, and efficiently. But today, a "software vs. people" showdown appears imminent. Software is increasingly a threat to people, organizations, and nations. For example, the spread of the Melissa virus illustrates the ease with which systems can be penetrated and the ubiquity of the consequences; it caused many companies to shut down their e-mail systems for days. The origin of these threats stems from a variety of problems. One problem is negligent development practices that lead to defective software. Security vulnerabilities that occur as a result of negligent development practices (such as commercial Web browsers allowing unauthorized people to access confidential data) are likely to be discovered by rogue individuals with malicious intentions. Other security vulnerabilities are deliberately programmed into software (logic bombs, Trojan Horses, Easter eggs). Regardless of why information systems are vulnerable, the end result can be disastrous and widespread.

Because of the increased danger that malicious software now poses, we seek original articles on the following topics:

- Intrusion detection
- Information survivability
- Federal critical-infrastructure protection plans
- Federal laws prohibiting encryption exports vs. US corporations
- State of the practice in security testing
- The Internet's "hacker underground"
- Corporate information insurance
- Penalties for those convicted of creating viruses
- Case studies in information security and survivability

Guest Editors:

Nancy Mead
Carnegie Mellon University
nrm@sei.cmu.edu

Jeffrey Voas
Reliable Software Technologies
jmvoas@rstcorp.com

Submissions due: 1 April 2000

Authors: Submit one electronic copy in RTF interchange or MS-Word format and one PostScript or PDF version to the magazine assistant at software@computer.org. Articles must not exceed 5,400 words including tables and figures, which count for 200 words each. For detailed author guidelines, see www.computer.org/software/edguide.htm. **Reviewers:** Please e-mail your contact information and areas of interest to a guest editor.

25Oct1999