Software Project Management

SEI Curriculum Module SEI-CM-21-1.0

July 1989

James E. Tomayko
The Wichita State University

Harvey K. Hallman
Software Engineering Institute

Carnegie Mellon University
Software Engineering Institute

This work was sponsored by the U.S. Department of Defense.

Draft For Public Review

The Software Engineering Institute (SEI) is a federally funded research and development center, operated by Carnegie Mellon University under contract with the United States Department of Defense.

The SEI Education Program is developing a wide range of materials to support software engineering education. A **curriculum module** identifies and outlines the content of a specific topic area, and is intended to be used by an instructor in *designing* a course. A **support materials** package includes materials helpful in *teaching* a course. Other materials under development include model curricula, textbooks, educational software, and a variety of reports and proceedings.

SEI educational materials are being made available to educators throughout the academic, industrial, and government communities. The use of these materials in a course does not in any way constitute an endorsement of the course by the SEI, by Carnegie Mellon University, or by the United States government.

SEI curriculum modules may be copied or incorporated into other materials, but not for profit, provided that appropriate credit is given to the SEI and to the original author of the materials.

Comments on SEI educational publications, reports concerning their use, and requests for additional information should be addressed to the Director of Education, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

Comments on this curriculum module may also be directed to the module authors.

James E. Tomayko
Computer Science Department
The Wichita State University
Wichita, KS 67208

Harvey K. Hallman
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Software Project Management

Acknowledgements

Rich Pethia, Lauren Roberts Gold, Betty Deimel, and Watts Humphrey of the SEI and Brad Brown of Boeing Military Airplanes made important suggestions for the content and organization of this module. Dick Fairley and John Brackett did helpful reviews at a critical stage in its development. Ron Wild, of Wichita State, helped implement revisions and maintain "requirements traceability." Jim Rankin, of Carnegie Mellon, carried on this work through the latter stages of editing. Jim Tomayko's students at Wichita State, who suffered through early use of this module, gave valuable feedback.

We would also like to thank Lionel Deimel and Linda Pesante of the SEI for technical, organizational, and stylistic assistance.

Contents

Software Project Management

Module Revision History

Version 1.0 (July 1989) Draft for public review

Software Project Management

Capsule Description

Software project management encompasses the knowledge, techniques, and tools necessary to manage the development of software products. This curriculum module discusses material that managers need to create a plan for software development, using effective estimation of size and effort, and to execute that plan with attention to productivity and quality. Within this context, topics such as risk management, alternative life-cycle models, development team organization, and management of technical people are also discussed.

Philosophy

Software project management remains different from project management in other, more established fields for a number of reasons: Software is a "brain product" only, unconstrained by the laws of physics or by the limits of manufacturing processes. It is difficult to detect and prevent defects in software. Software can be highly complex. Finally, as a discipline, software development is so young that measurable, effective techniques are not yet available, and those that are available are not well-calibrated. Despite these difficulties, there is an increasing body of knowledge about software project management. This module presents that knowledge and also points to promising new conceptual material.

The intent of the authors is for the material in this module to be useful in designing either a stand-alone graduate course on software project management or in preparing units of courses dealing with the early phases of the software development life cycle. Because of the level of experience of typical graduate students and their probable career paths, we emphasize methods, tools, and techniques sufficient to run small projects of 15 persons or fewer. Students may eventually run larger groups, consisting of teams of such teams, so some discussion of scaling up is included. Also, information about assessing the efficacy of the software development process is included.

We believe that software project management should be part of software engineering programs because the technology of developing software is so closely tied to the techniques of management. What is described here is first-line management; product management and higher levels of management are the domains of schools of business. Some of the material discussed here, such as scheduling tools like PERT and concepts like tracking progress, is also part of project management courses offered by business schools; but the point of this module is to consider such tools and concepts in the context of software development.

Objectives

The objectives for this module are presented in three groups—in a set of general objectives and in sets of more specific objectives in the cognitive and affective domains. The latter sets of objectives are useful in constructing specific courses or other units of instruction and in writing examinations for them.

General Objectives:

1. The student will understand the requirements for the content of a project management plan.
2. The student will be able to write a plan for a small project according to an established standard.
3. The student will understand the role of the manager in each phase of the software development life cycle.

Cognitive Domain Objectives:

At the completion of a course based on all the material in this module, a student will be able to:

1. Explain what aspects of software production are different from hardware production and what aspects are similar.
2. Define quality, productivity, and risk reduction within the context of effective software project management.
3. Describe, compare, and contrast at least two innately different models of software development resource estimation in the areas of: cost, personnel, physical resources, schedule, size of the product, and complexity of the product.
4. Explain the advantages and disadvantages of prototypes for risk management.
5. Define inspection technology and its application, and relate it to quality improvement.
6. List three examples of standards in different arenas, such as government, corporate, and within teams.
7. Define, compare, and contrast the following methods of project team organization: chief programmer, surgical, egoless, democratic, and hierarchical.
8. Describe methods of tracking factors such as effort and resources during the development process.
9. Define the objectives and context of the following formal reviews: requirements, preliminary design, critical design, and acceptance.
10. List and define three methods of progress reporting.
11. Define configuration management and describe its role in project management.
12. Define quality assurance and describe its role in project management.
13. List three factors that are prime motivators of software development personnel.
14. List three factors that demoralize software development personnel.
15. Identify two techniques useful in the assessment of completed a project.
16. List three special considerations in managing sustaining engineering efforts.
17. Explain how requirements specification influences the development of the project plan.
18. Describe at least two alternative life-cycle models and explain in what situations their use might be desirable.
19. List three techniques to restore a failing project.
20. List three indicators of the level of an organization's ability to perform the software development process.

Affective Domain Objectives:

1. The student will appreciate the key roles managers play in software development efforts.
2. The student will appreciate economic and customer-driven factors and their role in the eventual form of the software product.

Prerequisite Knowledge

Students should have prior experience as members of a software development team. If they do not have actual industrial experience, membership on a student project team of several persons can be sufficient if the team was large enough so the students can appreciate the necessity of good management principles and activities. Students should also have knowledge of the aspects of software development at a depth equal to that presented in a one-term introductory course on the subject.

Module Content

The scope of this curriculum module is unusually broad. In-depth coverage of all individual topics is not possible, nor is it possible to cite all important references. Wherever possible, the reader is referred to other SEI curriculum modules, both for more detailed topic discussion and for additional references. For convenience, cited curriculum modules and related SEI publications are listed separately in the bibliographies.

Outline

I. Introduction to Software Project Management
  1. The Nature of Software Production
  2. Key Objectives of Effective Management
    a. Quality
    b. Productivity
    c. Risk reduction
  3. The Role of the Software Project Manager

II. Planning the Project
  1. Business Planning
    a. Determining objectives
    b. Forecasting demand for the product
    c. Proposal writing
    d. Requirements analysis
    e. Legal issues (patent, copyright, liability, warranty)
  2. Technical Planning
    a. Life-cycle models
    b. Types of plans
    c. Plan documentation methods
      (i) Work breakdown structures
      (ii) PERT and CPM
      (iii) Gantt charts
      (iv) Standards
    d. Planning for risk management and control
      (i) Entry and exit criteria
      (ii) Intermediate checkpoints
      (iii) Performance prediction and analysis
      (iv) Prototyping and modeling
      (v) Inspections and reviews
      (vi) Process and process assessment
      (vii) Development methods
      (viii) Metrics
      (ix) Configuration management
      (x) Testing and quality assurance
      (xi) Capacity planning
    e. Estimating—what it takes to do the job
      (i) Cost (direct and indirect)
      (ii) Resources
      (iii) Time
      (iv) Size and complexity of the product
      (v) Risk determination
      (vi) Role of requirements and design in estimating
    f. Financial planning—budgeting
    g. Resource allocation
    h. Organizational considerations (teams, hierarchies, etc.)
    i. Technology
    j. Human factors and usability
    k. Tools and environments
    l. Transition of the product to the user

III. Managing the Project
  1. Managing the Task
    a. Project control
      (i) Managing to the plan
      (ii) Reviews
      (iii) Feedback and reporting mechanisms
      (iv) Configuration management
      (v) Quality control and quality assurance
      (vi) Managing change
      (vii) Readjusting goals and milestones
    b. Risk management
    c. Testing phases
    d. Formalized support activities
  2. Managing the Team
    a. Team organizations
    b. Recruiting and staffing—picking the right people
    c. Technical leadership

d. Avoiding obsolescence—training, etc.

3. Managing the Context
   a. Communications skill
   b. Decision theory
   c. Business management
   d. Assessing the organization's ability to perform the process
   e. Probability and statistics
4. Managing Product Support and Maintenance

IV. Evaluating the Project

Annotated Outline

I. Introduction to Software Project Management

1. The Nature of Software Production

Software project management is different from managing hardware-like projects because of the different nature of software itself. Software products are generally more complex than the hardware on which it runs. Frequently problems too complex or expensive to solve in the electronics are passed off to the supporting software—rotor position in a stepper motor, print wheel position on a line printer, position of a satellite relative to a star. There is basically no manufacturing process. The development process ends with delivery to the user. Problems normally discovered during manufacturing have to be found during development. Inspections (reviews) that are normally part of the manufacturing process have to be moved back into the development phase.

Since software frequently provides the interface to the user, any frustrations with the system tend to be focused on the software. As a result, there may be frequent changes in the requirements description. Since it appears to most people that it is easier to change the software than the hardware, most of these requirements changes end up being made in the software, which is often a mistake.

Large numbers of people, often geographically distributed, are needed to solve many of the more important problems that software is used to solve. This requires extensive communications mechanisms in order to complete the software product.

Many times software engineers lack the domain knowledge necessary to truly understand the requirements. There is often no way to tell how long it will take to acquire this knowledge. As a result, it is difficult to plan, estimate the size of, change, and produce software. To make matters worse, there is often little discipline in the development process and documentation associated with small software projects. Many large software projects are in the same state, but they are further hampered by the complexity of the organization, in addition to the complexity of the product. This is often true with large hardware projects, but software development has additional disadvantages. There are fewer tools available to help in software construction; and the profession has not matured to the same level as other engineering disciplines.

2. Key Objectives of Effective Management

A software development manager must keep in mind quality, productivity, and risk reduction throughout the planning and execution of product development. [Brooks75] is an outstanding overview of these concepts.

a. Quality

Quality is best achieved by careful adherence to standards, effective development techniques, and periodic technical review throughout the process. Management must cooperate and coordinate with quality assurance organizations, if they exist [IBM85, Deming86]. The Japanese emphasis on continuous quality improvement and its remarkable results offer important lessons for software developers [Mann88].

b. Productivity

Increased productivity lowers costs. In the current state of development technology, the most important productivity factors are the ability of the individual software engineers, the tools they work with, and the work environment [Boehm87, Weisbord88].

c. Risk reduction

Managers should identify the most difficult parts of a particular development and systematically come to grips with efficient solutions. Requirements that can become "show-stoppers" later must be dealt with early in the process. (See [Boehm88] for information on risk-reduction life cycles.)

3. The Role of the Software Project Manager

The role of the "manager" is to plan, organize, staff, direct, and control. He or she deals with ideas, things, and people. Software development, however, involves more "scheduled creativity" than most other fields of endeavor. The manager has to commit to delivery of copies of the product before the product has been completely developed. In most engineering professions, the creative development of the product occurs before the commitment to build or manufacture one or more copies.

The software project manager has to plan and organize activities of his or her staff in such a way that there is confidence that the promised end product will be delivered on time, within the cost projected, and with the quality expected. This is accomplished by assembling a staff that is qualified to do the tasks required, by directing and redirecting the staff as necessary, and by putting the controls in place that will provide the information necessary to direct the staff to its completed assignment. These controls are frequently referred to as risk management.

II. Planning the Project

When it has been determined that a software product will be developed, a project or other organizational entity is put in place to accomplish it. This project or entity should develop the necessary plans to accomplish the task.

Project planning can be divided into business plans, which concentrate on the relationship with the customer, and technical plans used internally in the development group.

1. Business Planning

a. Determining objectives

The function, cost, and price objectives for a software product need to be determined. Also, the objectives of the project developing the product need to be stated [Kotler88, Kerin87].

b. Forecasting demand for the product

In some environments, a software product is developed on speculation that there is a market for the product. (See [Radice88], pp. 60-68.) A product forecast is used to estimate the number of units that it will be possible to sell. It is a function of projected development cost, price of the potential product, and the time at which the product can be made available. In lieu of a contract to build a product for a specific organization, software development houses find it necessary to know what the potential market for the product may be before committing large amounts of venture resources to the project. Chapter 15 of [Anderson85] gives an overview of general forecasting methods.

c. Proposal writing

In many environments, a contract is sought with an organization or the government. This is usually initiated by a request for a proposal and price, sometimes called an RFP. The software organization may, by itself or in conjunction with another organization, prepare a proposal in the hopes of obtaining a contract for the work. This may be in the form of a "cost plus" contract, where the income from the contract is based on the cost to complete it plus an agreed-to profit. The contract may sometimes be a "fixed price" contract, where the developer agrees to do the work for a fixed amount of money. During the proposal writing process, much of the project planning will be accomplished or approximated. It is often completed or redone again after the contract is received.

d. Requirements analysis

It is important that the requirements be analyzed, so that it is understood what is needed from a functional characteristics point of view. This is discussed in the curriculum module, *Software Requirements* [Brackett88].

e. Legal issues (patent, copyright, liability, warranty)

Managers should have knowledge of whether algorithms qualify for patent or copyright protection. Managers should also verify that software meets liability and warranty requirements. Useful readings include [Chisum86], [Newell86], [Holmes82], [Friedman87], and [Cottrell86]. Also see *Intellectual Property Protection for Software* [Samuelson89] and *Software Development and Licensing Contracts* [Samuelson88].

2. Technical Planning

a. Life-cycle models

The planning process is dependent upon the life-cycle model used by the development organization.

Teaching Consideration: *It is important that the student be exposed to life-cycle models of software development before he or she gets very far into the planning process. This is necessary in order to best understand who does what. See* Models of Software Evolution: Life Cycle and Process *[Scacchi87] for an understanding of software life-cycle models.*

b. Types of plans

There are a number of plans developed in support of a large software project. These are described in chapter 7 of [Shere88]. They include:

- Program master plan
- Management plan
- Development plan
- Configuration management plan
- Quality assurance plan
- Maintenance plan
- Test plan
- Integration plan
- Documentation plan
- Transition plan
- Firmware development plan

c. Plan documentation methods

(i) Work breakdown structures

Work breakdown structures (WBS) are used to divide large projects into separate, manageable tasks [Shere88], similar to the way software is divided using stepwise refinement. They are sometimes used to break the work down organizationally [Evans83]. They are used throughout the planning process to refine the work structure and identify who does what. At the lowest level, they are used to identify the specific piece of work (work package) to be accomplished. They do not show sequence or time relationships. Determining the work breakdown structure for a project is discussed in [Tausworthe80].

(ii) PERT and CPM

Activity networks (PERT charts) and critical path methods (CPM) are used to describe dependencies, in flow diagram form, between the project parts. They identify the duration of an activity, the start and completion dates, and the critical path to getting a project completed. A critical path is the longest path in the network, thus delays to any activity on it delays completion of the project. [Cori85] is a good summary of how to implement these techniques. A deeper treatment of network models and the use of PERT/CPM models are discussed in Chapters 9 and 10 of [Anderson85].

(iii) Gantt charts

Gantt charts [Clark38] are similar in function to PERT and CPM but show the dependencies in bar chart form rather than flow form [Shere88]. The charts use elapsed time across the horizontal axes and activities along the vertical axes. They are especially useful in identifying who does what and determining whether all of the activities will be finished by certain dates. Although Gantt charts are not useful for showing interdependences, as are PERT charts, they do more easily show overlapping activities and individual responsibilities.

MacProject is a tool that illustrates PERT and Gantt charts [MacProject89].

(iv) Standards

When software organizations are fulfilling contracts, they are often required to follow the development standards of the originator of the contract. In addition, many companies have their own development standards that either supplement the contractor's requirements or apply to all software development groups in that company. Development groups also create standards to either supplement those already placed upon them or to establish an orderly development process. All of these standards have an effect on the planning process in the format of the plans and the sequence of activities to be accomplished.

DOD-STD-2167A [DoD88] is the software development standard for organizations contracting with the U. S. Department of Defense.

NASA-Sfw-DID-02-ADA [NASA86a] is the software management plan standard for organizations contracting with NASA.

IEEE Std 1058.1-1987 [IEEE87] is the IEEE statement for software project management plans for those organizations that do not have standards required of them by contract.

d. Planning for risk management and control

Risk management is the process of assuring that all problems are discovered early enough so that there is time to recover from the problem without missing schedules or overspending the budget. Control mechanisms are put in place in order for the feedback to occur at the proper time.

(i) Entry and exit criteria

All distinct phases of the project should have entry and exit criteria [Radice88]. These should be documented as part of the development plan. No phase should be started if the entry criteria have not been satisfied or the reasons for exceptions to the criteria are not known and agreed upon. No phase is complete until the exit criteria are satisfied. For example, coding of a module should not begin until the design of the module is complete. Exceptions are sometimes negotiated.

(ii) Intermediate checkpoints

Some activities in the planning process are aimed solely at reducing risk. Setting up a series of checkpoints throughout the planning period and treating each checkpoint as though it were an actual delivery date reduces the risks of missing the final delivery date, of burning people out, and of overrunning budget costs.

To accomplish this, checkpoints are established with intervals of four to six months, but at significant places in the plan, where real progress can be measured. With small projects, it may be that these significant checkpoints are established about one-fourth, one-half, and three-fourths of the way to the final delivery date.

For each checkpoint, a list of internal deliverables is established. All parts of the

project are managed so that completion of these deliverables occurs on or before the checkpoint date. This has the effect of creating a number of points where there is increased activity, rather than only one, at the end. This increased activity is of a lower intensity and is spread out over the project. Its intensity is felt only by those working extra time to meet the checkpoints. It also spreads the overtime required to resolve problems over the project, instead of requiring all of it at the end. Overtime in itself is a risk-taking activity (see below). When there is only one major date, the final delivery date, the intensity of activity comes to a higher peak; it is usually felt by everyone; and it often results in missing the final delivery date or delivering a lower quality product, or both. In general, the later problems are discovered, the harder they are to fix.

Overtime reduces the effectiveness of individuals on future work. Putting an individual on overtime for a short spurt can be effective in overcoming a problem. However, it reduces the team's ability to resolve problems yet to be discovered. The longer a person is on continuous overtime, the less effective he or she is. Moreover, it has been found that in areas where the effectiveness of an individual depends on his or her creative thinking and is not paced by a mechanical device, the productivity of an individual diminishes the longer that person is on overtime. That is, if you increase a person's working hours by two or three hours every day and continue this for a long period of time (several months), the person will become less effective than he or she was before the overtime started [Metzger87].

(iii) Performance prediction and analysis

When performance characteristics are included in the requirements specification, it may be necessary to develop a performance model in order to determine the possibility of meeting the requirements [Sauer81, Ferrari78, Shannon75]. It may also be necessary to analyze the performance of the product as it is developed, in order to ensure that predictions are being met. Occasionally, the characteristics of the hardware also have to be modeled [Lavenberg83]. The plan may be affected by these efforts.

(iv) Prototyping and modeling

During software product development, it is often discovered in the early stages that there is not enough confidence in the technology to be assured that it will work properly when it is completed. A prototype or working (sometimes simulation) model of the technology is developed to get a better understanding of the workability of the approach. The prototype is usually put aside, and the product is developed with the knowledge gained. In some cases, the prototype is not discarded but is evolved into the final product. The "spiral model" form of the development process [Boehm84a, Boehm-88] is based on this technique. Time must be allowed in the development plans for prototyping and modeling if the nature of the product requires it.

(v) Inspections and reviews

Inspections, walkthroughs, and reviews are techniques aimed at reducing the number of errors that pass into the next phase of the development process. These techniques are described in detail in the curriculum module *The Software Technical Review Process* [Collofello88b]. (See also [Cross88].) The more formal and most effective of these is the inspection process [Fagan76].

(vi) Process and process assessment

Every development project uses a process of some form. It may be unplanned or *ad hoc*. It may be repeatable or well-defined. It may even be well-managed and optimized [Humphrey88]. The process has a strong influence on the development plans. [Humphrey89] treats this topic in detail.

(vii) Development methods

It is essential that all software development projects follow some effective development method. The curriculum module *Introduction to Software Design* [Budgen89] has references to many of the more popular design methods. See also [Scacchi87] for additional information on life-cycle–specific methods.

(viii) Metrics

Software metrics are a major ingredient in the proper management of a software project. Without them, the manager has little knowledge of what the project is doing or how it will meet its commitments. The curriculum module *Software Metrics* [Mills88] addresses this subject in detail.

(ix) Configuration management

Software configuration management encompasses the disciplines and techniques of initiating, evaluating, and controlling change to software products. It emphasizes the importance of configuration control in managing software production. See *Software Configuration Management* [Tomayko87a].

Teaching Consideration: *Here the emphasis is on configuration management planning. Later (see III.1.a.iv), it is on implementation.*

(x) Testing and quality assurance

Even the most perfectly developed product needs to be tested. A software project manager needs to know the different types of testing and the purpose for each. *Introduction to Software Verification and Validation* [Collofello88a] provides an overview of testing concerns, while *Unit Testing and Analysis* [Morell89] provides a more detailed description of one aspect.

Testing activities are aimed at discovering errors in the product after it has been created, so that action can be taken to remove them. Quality assurance activities are aimed at preventing errors from getting into the product, keeping them from multiplying, and determining how many are in the product when it is delivered. (See *Quality control and quality assurance*, below.)

(xi) Capacity planning

Capacity planning is just as important in software development as it is in hardware manufacturing. Misjudging the organization's ability to conduct a project can be very costly. Additional people may have to be hired. This could require additional space and facilities. Computer resources may not be adequate for the additional responsibility. Section 12.1 in [Shere88] addresses this subject.

e. Estimating—what it takes to do the job

(i) Cost (direct and indirect)

Understanding the origin of software costs is an important first step in cost estimation [Brooks75].

The cost of developing a product contains many variables: the direct costs attributed to the effort spent directly on the project as well as the indirect costs that occur, such as office space, company benefits, etc. There have been many approaches to developing resource models over the years [Basili80]. Cost estimation models are gaining popularity. COCOMO, based on lines of code, is described in [Boehm81] and summarized in [Boehm84a]. Function points are described in [Albrecht83]. Modifications of function points have recently become available [Symons88]. [Pressman87] has clear introductions to how to use both function points and COCOMO. The chief difference between the two is that COCOMO is based on lines-of-code estimation, while function points methods depend on identifying specific functions within the software in order to calculate a number that indicates relative complexity. [Kemerer87] is a good evaluation of the effectiveness of the techniques.

(ii) Resources

It is necessary to determine what sort of computing and other physical resources are required in order to successfully complete a project. This activity includes decisions relating to development environments, which have a heavy influence on productivity. People-loading estimates are related to time and cost factors, and consist of a variety of tradeoffs. People-loading estimates can be made using COCOMO. Information on environments is in [Dart87].

(iii) Time

The direct relationship of time to cost and resources is best shown by analysis of data generated by using COCOMO. [Boehm81] gives many examples.

(iv) Size and complexity of the product

Cost estimation models depend on accurate size estimates. Estimating by analogy, using Putnam's model, conversion from function points, and wideband Delphi techniques are some size estimation methods [Putnam80].

(v) Risk determination

Failures in cost estimation can have drastic effects on the project. Identifying areas of possible risk and taking action to understand them can minimize their effects. For fallacies in estimation, see [Abdel-Hamid86] and [Kemerer-87].

(vi) Role of requirements and design in estimating

Most cost estimating techniques need a requirements document. Function points are based on an analysis of the requirements document. Other cost estimation techniques, such as COCOMO, depend on some level of design being complete. The design should be at least at a level from which lines-of-code estimates may be approximated.

f. Financial planning—budgeting

All projects have to be concerned with financial plans. These plans are concerned with assuring that the project has the funds to accomplish its objectives. The budget is tied to the cost estimation process. It is a plan that includes how the money will be spent and for what it has been al-

located. Personnel expenses, purchase orders, and computer expense accounts are some of the items charged against the budget. Other items, such as facilities and benefits, are usually accumulated into overhead charges and need to be considered. A variance report (usually available monthly) shows how the project is spending against the budget. A software project manager is involved in this financial process. This is where he or she gets the wherewithal to obtain the resources to complete the project. The process is usually unique to the company or organization to which the project belongs.

g. Resource allocation

Allocation depends on schedule, competition between projects for resources, use of workstations versus common computational resources, etc. This must be determined in advance and included in the project plan.

h. Organizational considerations (teams, hierarchies, etc.)

The way the development organization is set up has an effect on the cost estimates. See *Managing the Team* below, for details on approaches to organization.

i. Technology

Software project managers need to know more than just software development. In order to make intelligent decisions about the software product, they must understand the technology of the domain in which the software is to be used. Without this knowledge, they are dependent upon the knowledge of someone else who may not have the larger context needed to manage the project.

j. Human factors and usability

User interfaces and their associated human factors are so important to the success of a software product that the software project manager should be aware of their significance and be prepared to ask the appropriate questions of his or her development team. [Radice88]. See *User Interface Development* [Perlman88].

k. Tools and environments

Tools and environments are essential to the software manager on a limited budget. In the ’50s and ’60s, language processors, editors, operating systems, and generators were the tools of concern. In the ’70s, it was terminals as input devices, inspections, document and network analyzers. In the ’80s, more modern tools are of concern: process-oriented tools such as configuration managers, syntax-directed editors, workstations, and automated environments that cover the entire development process [Radice85].

l. Transition of the product to the user

Whenever a software product is to be used outside of the organization that developed it, some preparation has to be made for its use. This includes more than the availability of user manuals. Often a transition plan is developed, defining the work that has to be done to get the product used by the customers. Larger products may have to be given early to a few potential customers (through beta tests or early support programs) so that the transition plan can be tested.

Transition plans may include preparation of special documents on how to get the system up and running; assignment of personnel to support the installation of the product and the training of these installers; teaching of classes for the users or the trainers of the users; or documentation of the product that has to be included in contract software so that the customer can maintain it. It may even include the development of a sales plan for marketing the product.

Pressman discusses at length the issues associated with the introduction of new technology in [Pressman88].

III. Managing the Project

1. Managing the Task

The objective of process management is to assure that the project is completed on time, within budget, and with high quality.

a. Project control

(i) Managing to the plan

(1) Tracking events

Tracking can be accomplished through regularly scheduled status review meetings and through status reports at intervals less than the scheduled completion time of work packages.

(2) Tracking requirements

Tracking requirements are sometimes required by contract. In any case, it is advisable to develop a method for tracking requirements in the work products when the project is large and subject to requirements changes. See *Software Requirements* [Brackett88].

(ii) Reviews

Reviews are an important part of a formal quality assurance process. See *The Software Technical Review Process* [Collofello88b] for more information.

(iii) Feedback and reporting mechanisms

Generating feedback from tracking and review activities is essential, otherwise the information useful to the manager cannot be obtained. This is further discussed in [Brooks75], chapters 6, 7, and 14. Information on project audits is in [Bernstein81].

(iv) Configuration management

Managers should be aware of the methods for implementing software configuration management. See *Software Configuration Management* [Tomayko87a].

(v) Quality control and quality assurance

Software quality assurance is defined in the *IEEE Standard for Software Quality Assurance Plans* [IEEE84] and again in *IEEE Standard Glossary of Software Engineering Terminology* [IEEE83]. Under these definitions, SQA is a "planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements."

There are a number of interpretations of what should be included under the umbrella of quality assurance. In some organizations, QA manifests itself as an independent organization that audits the work of the development organization more in the manner that quality control is used in manufacturing. This may include the running of tests of the software products [Fairley85].

In other organizations, QA takes on all activities that can effect the quality of the product, including configuration management and reviews. Brad Brown's curriculum module *Assurance of Software Quality* [Brown87] takes this approach. The software project manager should be aware that both definitions exist. [Buckley84] is an introduction to the subject. [Basili87] discusses the "cleanroom" approach to emphasize quality in software development.

(vi) Managing change

One experienced software manager, Richard Parten, said when he was at NASA that "project management is change management." Both changes to the software product [Harvey86] and to the development environment (including personnel) must be expected and planned for. If these changes occur without being planned for, chaos will result and failure to meet schedules will be ensured. Any of Bersoff's writings are useful background [Bersoff84, Bersoff80].

(vii) Readjusting goals and milestones

Information gained from status meetings can be assessed and then used to adjust schedules and milestones. These should be reflected in the planning documents, such as PERT/Gantt charts and other scheduling materials, for tracking.

b. Risk management

The manager has to act on the results and data provided by the risk management activities established during the planning process, such as prototyping, performance modeling, reviews, etc.

c. Testing phases

The testing stages in software development need special attention. Delays at this time in the development cycle may cause delays in the delivery of the product or may negatively effect the reputation of the development team. See *Introduction to Software Verification and Validation* [Collofello88a] for more details on testing and error prevention techniques. For formal techniques to prevent errors, see *Formal Verification of Programs* [Berztiss88].

d. Formalized support activities

Freeman proposes that "a formal support activity is an essential part of a modern SDS" (software development system) [Freeman87]. This is true not only for modern development tools but for all development tools. Brooks refers to the persons responsible for support as the "language lawyer" and the "toolsmith" in his surgical team concept [Brooks75]. It is essential that the development organization have access to the support of experts who know the tools well. It is not a luxury. It can be as simple as assigning the responsibility for the tool to one of the developers, as happens in many small organizations, or having the tool developers provide constantly available support of the developers' use of the tools, as happens in many large organizations.

There is massive evidence that inadequate tool support costs valuable time and increases the errors in the software. Any first-line manager or team leader can attest to the time lost from improper support of the tools. Freeman claims that 10 percent of the project's resources devoted to this "support activity is not uncommon in sophisticated organizations that work on complex systems."

2. Managing the Team

Effective people management in software projects takes into account the differences in personality among the various members of the team and how

their roles affect their personalities. [Metzger87] is an easy-to-read source of material that does not go too deeply into the topic for students. DeMarco and Lister have also written about software teams and projects [DeMarco87].

a. Team organizations

There are many types of team organizations. Mills proposes the use of chief programmer teams [Mills83]. Brooks proposes surgical teams in [Brooks75], chapters 3 and 7. Weinberg promotes the egoless approach [Weinberg84], which is similar to the democratic approach described by Brooks. Stuckenbruck describes a matrix organization [Stuckenbruck81]. Mantei discusses the effects of various team structures [Mantei81]. Weisbord presents an approach to designing and managing work teams that promotes greater productivity [Weisbord88].

b. Recruiting and staffing—picking the right people

Frequently part of the software project manager's job is to get the proper people together to do the project. This may mean borrowing the people from another organization as is done in matrix organizations. But more often it means recruiting the people needed from either inside or outside the organization.

Recruiting from inside the organization occurs when new work is taken on to use the people who become available as other projects near completion. As projects complete, they begin to staff down and make their people available. It is the new project manager's challenge to find the proper people to do the new project and convince the individuals that this project is the right place for them.

The process is not unlike recruiting new employees. However, the manager knows the planned availability of people, not by individuals but by numbers. He or she also knows what projects will be reducing staff. It is the new project manager's job to negotiate with the existing project managers. Since it is the objective of the new project to staff with only the best people and the objective of the existing project manager to place the poorer people first (he or she wants to keep the more productive), there is a conflict of interest between the two. Much negotiation and compromise is required to reach agreement. Appeals to higher authority are frequently necessary. Negotiation is usually easier in large organizations than in small ones, as there are more recruiting options open to the new project manager.

Once agreement is reached, the new project manager has to interview the candidate, convince the person that the new project is right for his or her career, and negotiate a starting date. Much of this is done in parallel for several candidates for each position. These newly recruited people are able to become productive more quickly than those hired from outside the organization—they know the systems and standards used by the organization.

The problem with internal recruiting is that often the new manager does not have complete control over the quality of the individuals recruited. Successful internal recruiting depends on the priority of the project, the availability of good people, his or her negotiating ability, and the support given by upper management.

Recruiting from outside the organization is more difficult. First, it is necessary to do something to get people to apply for the job, such as advertise, hire a recruiter, etc. The manager has to screen applicants, bring the more promising ones in for an interview, negotiate salary, give an offer to the ones selected, start over when turned down, and provide additional training to the ones who accept. All this has to be accomplished before the new person is able to be productive on the new project. The result is that the manager takes longer to staff up but has more control over the quality of the individuals recruited. Success depends on how particular or anxious he or she is in filling the open positions. The manager trades off availability of high-quality people against project schedules and has to live with the consequences.

No matter how the staff is recruited, the new manager seldom has the ideal staff for the job. He or she needs to find the strengths and weaknesses of the new people and adapt the assignments to the people. Occasionally, a new person will be too weak in the area assigned and this will not be evident until the first or second review. Often the new person is a surprise and is better than was expected. The manager has to be prepared to readjust assignments in the middle of the project in order to get the best out of his or her staff.

c. Technical leadership

The attitudes of the members of the development team are an influencing factor in their performance from both a quality and productivity point of view. The motivation of individual team members is easily influenced both positively and negatively by the manager [Humphrey87].

See [Humphrey87] and [Weinberg86] for well-presented discussions of this topic. Some key points from [Humphrey87] are reproduced here:

"The technical leader's most important role is to set goals and drive unswerving to meet them."

"When one person makes a pact with another and they both expect it to be kept, that is a commitment. When technical groups must coordinate their efforts to produce a coherent result, schedules are needed and must be based on mutual agreements or commitments. Unfortunately, people rarely make accurate estimates, and something unexpected always comes up to delay things. Everyone has to scramble to meet the schedule. Why does everyone scramble instead of accepting the delay and blaming bad luck? The difference is the attitude of commitment."

"The two key elements of professionalism are the knowledge of what to do and the discipline to do it. Technical knowledge is the true mark of the professional for it sets him above his less learned friends."

"Respect for the individual rests on an attitude of fairness. Each person must be valued as an individual and treated according to his personal wants and needs."

d. Avoiding obsolescence—training, etc.

Continual upgrading of skills and background knowledge is essential in the quickly changing field of software development. The responsibility is partly the individual's and partly the manager's.

The manager is responsible for the training necessary to do the current job and for encouraging activities that will prepare the individual for the future. The manager needs to provide training events and experiences at the proper time, so the individuals can accomplish their tasks properly. This might simply provide the knowledge of how to use the tools or it may be a class on a new language. The manager also needs to expose the individuals to current developments in the field. He or she has to ensure that an up-to-date library is available with periodicals for the software profession and associated professions. The manager should encourage access to the advanced technology groups in the organization and attendance at conferences on subjects in the professional's area of expertise, or should bring in experts to give presentations on subjects of interest.

The individual also has considerable responsibility for the prevention of obsolescence. He or she can join professional societies such as ACM and IEEE. He or she needs to read the journals, browse through the library, and be aware of the current trends in technology being published.

A case study of the need for training and its effectiveness is contained in [Cupello88].

3. Managing the Context

In addition to the day-to-day, project-to-project task and team management concerns, software managers should have knowledge of the following topics related to improving their overall skills and their organization's capability.

a. Communications skill

The fundamentals of technical communications should be understood and practiced by the software project manager in both written and oral form. *An Overview of Technical Communication for the Software Engineer* [Glass88] provides some general information and a bibliography of other sources.

b. Decision theory

Decision theory can be used to develop optimal strategies when the decision maker is faced with several alternatives and the uncertainty of future events. The software development process is filled with many risk situations where the outcome is not necessarily obvious. Sometimes the design of a product is not straightforward. Usually, the testing outcome is unknown. Occasionally, alternative approaches are proposed. The concepts of payoff tables, decision making with uncertainty, decision trees, and expected values are a few of the approaches discussed in chapter 16 of [Anderson 85]. Sometimes it is simply that something that has been working ceases to work properly. Some solution approaches look at the problem of determining what is different or what is changed [Kepner65].

Management science is a broad discipline that deals with approaches to managerial decision making that are based upon the application of scientific methods, many of which are appropriate in the software engineering domain. It includes both quantitative and intuitive approaches.

c. Business management

Not covered in this module are all of the business management processes normally applicable in general management areas. These are usually addressed by the organization for which the manager works as company policy and standards. The topic of organization theory is covered in [Scott76] and [Jackson82].

d. Assessing the organization's ability to perform the process

Process evaluation is described in [Humphrey88]. This includes self-assessment of the level of sophistication of a particular organization's software development capabilities.

e. Probability and statistics

A knowledge of probability and statistics is required to perform many of the estimating and scheduling tasks. The mathematician requires that all data be available before a decision can be made [Feller68, Walpole85]. For the software manager who seldom has all of the data, a knowledge of Bayesian statistics is helpful [Berger85, Cyert87].

4. Managing Product Support and Maintenance

Managing maintenance and product support personnel is a serious motivational challenge. Also, reducing the cost of maintenance often requires "reverse engineering" of missing documents or comments in code. An interesting reading on this is [Collofello87], and there are several books on the subject, such as [Glass81] and [Arthur88].

IV. Evaluating the Project

A project history, which assesses the effectiveness of the estimates and project plan, is an increasingly important part of product development, even to the point of being included in standards (such as the NASA standard for software development [NASA86b]). It is such histories that form the core of corporate experience, making possible increasingly accurate application of models like COCOMO and function points, and providing a hedge against turnover of personnel.

Teaching Considerations

Textbooks

There is no single textbook that provides a comprehensive treatment of the topics in this module. Therefore, it is recommended that a mix of texts and journal articles be used to gain sufficient coverage of the issues. Many relevant articles are reprinted in the *IEEE Tutorial on Software Engineering Project Management* [Thayer88]. Some of them can be used to flesh out either key or ignored topics in texts such as [Gilb88] or [DeMarco82]. One book that does treat most topics is [Humphrey89].

Teaching Techniques

There are at least two approaches to the overall design of a software project management course. The first is sequential topic presentations as found in most course plans. The second is a spiral approach pioneered at the Wang Institute and described in [Spicer83].

The fundamental question is whether to include some actual project experience during the course. Spicer did a graduated set of projects: individual (lasting a week), small (a couple weeks), and medium (several weeks). Other approaches are to "farm out" the students to concurrently running software engineering project courses as team leaders, so they can obtain on-the-job training of a sort at the same time the material is being taught in class. When this course is taught in the context of a graduate level software engineering program, the master's project requirements will often provide management opportunities, so none are needed in the course itself. However, for the immediate future and probably indefinitely at schools enrolling significant numbers of part-time students, the experience of using tools and writing a project plan will have to come from within the project management course itself. There are again two approaches to this: either the students individually write a plan based on a requirements document provided by the instructor or the students are grouped into teams for developing the plan. Either way, a further decision has to be made about whether to make sections of the plan due at intervals or all at once.

Management topics should also be inserted in introductory software engineering courses. Typically, four weeks of such courses should be devoted to organization models, estimation, management plans, configuration management, quality assurance, and tracking. Cost accounting can be easily made a part of such courses, as demonstrated in [Tomayko87b].

SEI Video Course

The Software Engineering Institute Video Dissemination Project offers a complete videotape course on software project management. It includes lectures by several instructors and topic experts, plus support materials, including viewgraphs, reading list, exercises, and examinations. For additional information on this course, including its current cost, please contact the Technical Project Administrator of the Video Dissemination Project at the SEI.

Software Support

Software support for project management can be provided by automated environments and specific tools for scheduling and cost estimation. For example, the EPOS environment [EPOS88] provides a set of tools for project management that automatically generates such aids as Gantt charts. Individual tools, such as MacProject II [MacProject89], focus on activity networks and resource allocation. A microcomputer implementation of COCOMO developed at the Wang Institute, WICOMO, has been used successfully as part of the SEI video course mentioned above.

Bibliographies

Because of the broad scope of this curriculum module, it has been necessary to restrict the bibliography below to references concerned directly with software project management. Additional references on related topics can be found in the bibliographies of other SEI curriculum modules listed in a separate bibliography that follows.

Software Project Management References

Abdel-Hamid86

Abdel-Hamid, Tarek K., and Stuart E. Madnick. "Impact of Schedule Estimation on Software Project Behavior." *IEEE Software 3*, 4 (July 1986), 70-75.

> The thesis of this paper is that different estimates create different projects. Schedule estimates have impact on the progress of a project, in that they directly affect staffing, training, and perceived project status. Changes in these factors due to the estimates (reduction in personnel, shifting perceptions, etc.) can backfire.

Albrecht83

Albrecht, Allan J., and John E. Gaffney, Jr. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation." *IEEE Trans. Software Eng. SE-9*, 6 (Nov. 1983), 639-648.

> ***Abstract:*** *One of the most important problems faced by software developers and users is the prediction of the size of a programming system and its development effort. As an alternative to "size," one might deal with a measure of the "function" that the software is to perform. Albrecht [1] has developed a methodology to estimate the amount of the "function" the software is to perform, in terms of the data it is to use (absorb) and to generate (produce). The "function" is quantified as "function points," essentially, a weighted sum of the numbers of "inputs," "outputs," "master files," and "inquiries" provided to, or generated by, the software. This paper demonstrates the equivalence between Albrecht's external input/output data flow representation of a program (the "function points" metric) and Halstead's [2] "software science" or "software linguistics" model of a program as well as the "soft content" variation of Halstead's model suggested by Gaffney [7].*
>
> *Further, the high degree of correlation between "function points" and the eventual "SLOC" (source lines of code) of the program, and between "function points" and the work-effort required to develop the code, is demonstrated. The "function point" measure is thought to be more useful than "SLOC" as a prediction of work effort because "function points" are relatively easily estimated from a statement of basic requirements for a program early in the development cycle.*
>
> *The strong degree of equivalency between "function points" and "SLOC" shown in the paper suggests a two-step work-effort validation procedure, first using "function points" to estimate "SLOC," and then using "SLOC" to estimate the work-effort. This approach would provide validation of application development work plans and work-effort estimates early in the development cycle. The approach would also more effectively use the existing base of knowledge on producing "SLOC" until a similar base is developed for "function points."*
>
> *This paper assumes that the reader is familiar with the fundamental theory of "software science" measurements and the practice of validating estimates of work-effort to design and implement software applications (programs). If not, a review of [1]-[3] is suggested.*
>
> This paper is one of the early presentations of the "function points" method of cost estimation critically evaluated in [Kemerer87].

Anderson85

Anderson, D. R., D. J. Sweeney, and T. A. Williams. *An Introduction to Management Science, 4th Ed*. St. Paul: West Publishing Company, 1985.

> An excellent text used by many management science courses dealing with the "quantitative approaches to decision making."

Arthur88

Arthur, Lowell Jay. *Software Evolution*. New York: John Wiley and Sons, 1988.

> This book is a detailed survey of techniques for software maintenance activities. Its final chapter treats "managing for maintenance," and is good background reading for the topic when it is taught in a software project management course.

Basili80

Basili, Victor R. "Resource Models." In *Tutorial on Models and Metrics for Software Management and*

Engineering, Victor R. Basili, ed. New York: IEEE Computer Society Press, 1980, 4-9.

Describes the evolution of software development resource models. It divides the models into static, single-variable models; static, multivariable models; dynamic, multivariable models; and single-variable, theoretical models. The papers documenting the various models cited are included in the tutorial.

Basili87

Basili, Victor R., Richard W. Selby, and F. Terry Baker. "Cleanroom Software Development: An Empirical Evaluation." *IEEE Trans. Software Eng. SE-13*, 9 (Sept. 1987), 1027-1037.

Abstract: *The Cleanroom software development approach is intended to produce highly reliable software by integrating formal methods for specification and design, nonexecution-based program development, and statistically based independent testing. In an empirical study, 15 three-person teams developed versions of the same software system (800-2300 source lines); ten teams applied Cleanroom, while five applied a more traditional approach. This analysis characterizes the effect of Cleanroom on the delivered product, the software development process, and the developers.*

The major results of this study are the following. 1) Most of the developers were able to apply the techniques of Cleanroom effectively (six of the ten Cleanroom teams delivered at least 91 percent of the required system functions). 2) The Cleanroom teams' products met system requirements more completely and had a higher percentage of successful operationally generated test cases. 3) The source code developed using Cleanroom had more comments and less dense control-flow complexity. 4) The more successful Cleanroom developers modified their use of the implementation language; they used more procedure calls and IF statements, used fewer CASE and WHILE statements, and had a lower frequency of variable reuse (average number of occurrences per variable). 5) All ten Cleanroom teams made all of their scheduled intermediate product deliveries, while only two of the five non-Cleanroom teams did. 6) Although 86 percent of the Cleanroom developers indicated that they missed the satisfaction of program execution to some extent, this had no relation to the product quality measures of implementation completeness and successful operational tests. 7) Eighty-one percent of the Cleanroom developers said that they would use the approach again.

This paper can be used to illustrate quality-based life cycles.

Behrens83

Behrens, Charles A. "Measuring the Productivity of Computer Systems Development Activities with Function Points." *IEEE Trans. Software Eng. SE-9*, 6 (Nov. 1983), 648-652.

Abstract: *The function point method of measuring application development productivity developed by Albrecht is reviewed and a productivity improvement measure introduced. The measurement methodology is then applied to 24 development projects. Size, environment, and language effects on productivity are examined. The concept of a productivity index which removes size effects is defined and an analysis of the statistical significance of results is presented.*

This paper is a report of an application of the method described in [Albrecht83].

Berger85

Berger, J. O. *Statistical Decision Theory and Bayesian Analysis, 2nd Ed*. New York: Springer-Verlag, 1985.

Covers the foundations and concepts of statistical decision theory, including situations where data are incomplete. This book approaches statistics from the business perspective where not all of the data are available. It introduces the probability of accuracy of the data into the statistical calculations, statistics based on probable accuracy of the data (Bayesian), is not appreciated by the mathematical approach to statistics.

Bernstein81

Bernstein, L. "Software Project Management Audits." *J. Syst. and Software 2*, 4 (Dec. 1981), 281-287.

Abstract: *This paper shows how project audits can be used to uncover project strengths and weaknesses. Three audits are described and findings of the audit teams are summarized. Audits helped identify organizational problems, lacking management discipline, and software testing approaches useful to other projects. The issue of product sales versus disciplined project management was faced in all three audits. How this issue was handled is discussed and related to the success or lack of it for each project.*

This paper illustrates one method of status evaluation.

Bersoff80

Bersoff, E. H., V. D. Henderson, and S. G. Siegel. *Software Configuration Management*. Englewood Cliffs, N.J.: Prentice-Hall, 1980.

This book contains the most complete description of software configuration management available. It provides a fairly complete rationale for what to do and why. The authors have their own conceptual breakdown of the subject that does not map one-for-one onto the organization of this module. The book also does not clearly explain how to perform configuration management tasks.

Bersoff84

Bersoff, Edward H. "Elements of Software Configuration Management." *IEEE Trans. Software Eng. SE-10*, 1 (Jan. 1984), 79-87.

Abstract: *Software configuration management (SCM) is one of the disciplines of the 1980's which grew in response to the many failures of the software industry throughout the 1970's. Over the last ten years, computers have been applied to the solution of so many complex problems that our ability to manage these applications has all too frequently failed. This has resulted in the development of a series of "new" disciplines intended to help control the software process.*

This paper will focus on the discipline of SCM by first placing it in its proper context with respect to the rest of the software development process, as well as to the goals of that process. It will examine the constituent components of SCM, dwelling at some length on one of those components, configuration control. It will conclude with a look at what the 1980's might have in store.

This paper is good background reading for a lecture on SCM.

Boehm81

Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

This book *must* be read by every teacher or potential teacher of project management. Since economic matters are intricately involved in most aspects of software project management, the bulk of this book is usable in a course on that subject. Chapters 1-3 and 33 should be read first, followed by 4-9. These chapters provide basic background knowledge essential to every project management teacher.

Boehm84a

Boehm, Barry W. "Software Engineering Economics." *IEEE Trans. Software Eng. SE-10*, 1 (Jan. 1984), 4-21.

Abstract: *This paper summarizes the current state of the art and recent trends in software engineering economics. It provides an overview of economic analysis techniques and their applicability to software engineering and management. It surveys the field of software cost estimation, including the major estimation techniques available, the state of the art in algorithmic cost models, and the outstanding research issues in software cost estimation.*

This is a short summary of the thesis of [Boehm81].

Boehm84b

Boehm, Barry W., Terence E. Gray, and Thomas Seewaldt. "Prototyping Versus Specifying: A Multiproject Experiment." *IEEE Trans. Software Eng. SE-10*, 3 (May 1984), 290-302.

Abstract: *In this experiment, seven software teams developed versions of the same small-size (2000-4000 source instruction) application software product. Four teams used the Specifying approach. Three teams used the Prototyping approach.*

The main results of the experiment were the following.

1) *Prototyping yielded products with roughly equivalent performance, but with about 40 percent less code and 45 percent less effort.*
2) *The prototyped products rated somewhat lower on functionality and robustness, but higher on ease of use and ease of learning.*
3) *Specifying produced more coherent designs and software that was easier to integrate.*

The paper presents the experimental data supporting these and a number of additional conclusions.

This paper provides an example of an alternative development cycle.

Boehm87

Boehm, Barry W. "Improving Software Productivity." *Computer 20*, 9 (Sept. 1987), 43-57.

This article is an excellent short summary of the realistic factors involved in increasing the productivity of software developers. Boehm's explicit belief is that the quality of management is the most important factor in the success of a project. An extensive selected bibliography by productivity factor ("getting the best from people," "eliminating rework," etc.) is included.

Boehm88

Boehm, Barry W. "A Spiral Model of Software Development and Enhancement." *Computer 21*, 5 (May 1988), 61-72.

Yet another example of a risk-reduction life cycle.

Branstad84

Branstad, Martha, and Patricia B. Powell. "Software Engineering Project Standards." *IEEE Trans. Software Eng. SE-10*, 1 (Jan. 1984), 73-78.

Abstract: *Software Engineering Project Standards (SEPS) and their importance are presented in this paper by looking at standards in general, then progressively narrowing the view to software standards, to software engineering standards, and finally to SEPS. After defining SEPS, issues associated with the selection, support, and use of SEPS are examined and trends are discussed. A brief overview of existing software engineering standards is presented as the Appendix.*

This paper is useful as an overview if no specific standard is used in class.

Brooks75

Brooks, Frederick. *The Mythical Man-Month: Essays on Software Engineering*. Reading, Mass.: Addison-Wesley, 1975. The book was "reprinted with corrections" in January 1982.

Brooks wrote a true classic in the field of software engineering. Most of the text can be mined for management advice, as it is a "lessons learned" document of Brooks's career with IBM.

Brooks87

Brooks, Frederick P., Jr. "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer 24*, 4 (April 1987), 10-19.

In this article Brooks discusses why software isn't improving by leaps and bounds like hardware, why it never will, and what it takes to get the most out of software development.

Bryan84

Bryan, William, and Stanley Siegel. "Making Software Visible, Operational, and Maintainable in a Small Project Environment." *IEEE Trans. Software Eng. SE-10*, 1 (Jan. 1984), 59-67.

Abstract: *Practical suggestions are presented for effectively managing software development in small-project environments (i.e., no more than several million dollars per year). The suggestions are based on an approach to product development using a product assurance group that is independent from the development group. Within this check-and-balance management/development/product assurance structure, a design review process is described that effects an orderly transition from customer needs statement to software code. The testing activity that follows this process is then explained. Finally, the activities of a change control body (called a configuration control board) and supporting functions geared to maintaining delivered software are described. The suggested software management practices result from the experience of a small (approximately 100 employees) software engineering company that develops and maintains computer systems supporting real-time interactive commercial, industrial, and military applications.*

This paper can be used as an illustration of alternative quality assurance techniques.

Buckley84

Buckley, Fletcher J., and Robert Poston. "Software Quality Assurance." *IEEE Trans. Software Eng. SE-10*, 1 (Jan. 1984), 36-41.

Abstract: *This paper describes the status of software quality assurance as a relatively new and autonomous field. The history of its development from hardware quality assurance programs is discussed, current methods are reviewed, and future directions are indicated.*

This paper provides useful information for introducing quality assurance.

Cave78

Cave, William C., and Alan B. Salisbury. "Controlling the Software Life Cycle—The Project Management Task." *IEEE Trans. Software Eng. SE-4*, 4 (July 1978), 326-334.

Abstract: *This paper describes project management methods used for controlling the life cycle of large-scale software systems deployed in multiple installations over a wide geographic area. A set of management milestones is offered along with requirements and techniques for establishing and maintaining control of the project. In particular, a quantitative measure of software quality is proposed based upon functional value, availability, and maintenance costs. Conclusions drawn are based on the study of several cases and are generally applicable to both commercial and military systems.*

This paper is a useful overview of the tools, techniques, and approaches to managing software development.

Chisum86

Chisum, Donald S. "The Patentability of Algorithms." *U. of Pittsburgh Law Review 47*, 4 (Summer 1986), 959-1022.

Abstract: *In this Article, the author argues that algorithms, which are sets of specific rules for solving a mathematical or other type of problem and which are highly useful as the backbones of computer programs, should be the subject of patent protection if the patent law standards of novelty and unobviousness are met. After discussing case law*

developments prior to 1972, the author analyzes and criticizes the Supreme Court decision in Gottschalk v. Benson, *which held that mathematical algorithms were unpatentable per se as laws of nature or mere ideas. He argues that the* Benson *rule is unsupported by statute or case law authority or by policy considerations. He then analyzes the post-*Benson *Supreme Court decisions in* Parker v. Flook, *which extended* Benson, *and* Diamond v. Diehr, *which restricted it, and four 1982 decisions by the Court of Customs and Patent Appeals. He shows that theses cases fail to provide clear guidance on what types of algorithms may be patented as processes or methods. He contends that* Benson *is inconsistent in philosophy with the Supreme Court's 1980 decision in* Diamond v. Chakrabarty, *which held that genetically altered microorganisms may be patented and stressed that new technological developments covered by the general terms of the patent statute should be included in the patent system unless specifically excluded by Congress. The author concludes that patent protection for algorithms may be needed to provide incentives for innovation in software development as a supplement to copyright and trade secret protection of computer programs since the latter two forms of protection do not protect against unauthorized copying of publicly-disclosed ideas. A "computer scientist's response" on the issue of patentability and algorithms, by Professor Allen Newell of Carnegie Mellon University, follows this Article.*

This paper discusses the legal history of software, as well as the problems encountered in categorizing software for legal protection purposes.

Clark38

Clark, W. *The Gantt Chart*. London: Sir Isaac Pitman & Sons, 1938.

This book describes the various uses of bar charts following the ideas of Henry Gantt.

Collofello87

Collofello, James S., and Jeffrey J. Buck. "Software Quality Assurance for Maintenance." *IEEE Software 4*, 9 (Sept. 1987), 46-51.

Collofello and Buck provide some insight on managing a project for prevention of software problems, as well as correction of problems through software quality assurance.

Conte86

Conte, S. D., H. E. Dunsmore, and V. Y. Shen. *Software Engineering Metrics and Models*. Menlo Park, Calif.: Benjamin/Cummings, 1986.

This is the single most complete treatment of metrics and models usable in software engineering available. For project management, chapters 1, 2, 5, 6, 7, and 8 are most useful, but every project manager and instructor will need this book in his or her library.

Cooper78

Cooper, John D. "Corporate Level Software Management." *IEEE Trans. Software Eng. SE-4*, 4 (July 1978), 319.

Abstract: *Software management is considered from the corporate headquarters viewpoint. This perspective encompasses all facets of management, but specifically dealt with are those that are brought to bear on software management obstacles and ways to cope with them are presented. Standardization is presented as the most effective management device available at the corporate level for enhancing the overall software posture. Corporate management actions available for favorably influencing the quality of software over its life cycle and research initiatives are described.*

This is a much needed article addressing software management from the "other side" of the software development environment. Problems unique to software management are presented.

Cooper84

Cooper, Jack. "Software Development Management Planning." *IEEE Trans. Software Eng. SE-10,* 1 (Jan. 1984), 22-26.

Abstract: *The lack of comprehensive planning prior to the initiation of a software development project is a very pervasive failing. This paper walks through a sample software development plan discussing the various areas that a software development manager should address in preparing his project's plan. Various considerations and suggestions are presented for each of the management subject areas. How the user/customer can use the developer's plan to aid in monitoring of his software's evolution is also presented. Detailed planning of a software development project is necessary to the successful completion of the project.*

The step-by-step analysis of a sample software development plan in this paper forces many details to light that are ignored by other authors.

Cori85

Cori, Kent A. "Fundamentals of Master Scheduling for the Project Manager." *Project Management J. 16*, 2 (June 1985), 78-89.

This article outlines the functions and responsibilities of the project manager and surveys five different scheduling techniques. The material on milestones is particularly useful.

Cottrell86

Cottrell, Paul, and James Maron. "Professional Liability for Computer Design." *The Computer Lawyer 3*, 8 (Aug. 1986), 14-20.

This is a discussion of legal liability and responsibility for the computer industry.

Crawford85

Crawford, Stewart G., and M. Hosein Fallah. "Software Development Process Audits—A General Procedure." *Proc. 8th Intl. Conf. on Software Engineering.* Washington, D.C.: IEEE Computer Society Press, 1985, 137-141.

Abstract: *To assist development organizations in improving software quality and productivity, the AT&T Bell Laboratories Quality Assurance Center created a Design Quality Group to independently evaluate the software development processes and associated development products of our AT&T projects. These software development process audits examine software engineering techniques and tools in practice, as they fit into the overall development environment. Our strategy behind these audits is to assemble a team which, with the involvement of the developers and their managers, will: characterize the existing development process; identify project strengths and areas for improvements; and recommend possible improvements. This paper details the general approach behind this strategy.*

This is a good example for discussing the pros and cons of having an external quality assurance group.

Cupello88

Cupello, James M., and David J. Mishelevich. "Managing Prototype Knowledge/Expert System Projects." *Comm. ACM 31*, 5 (May 1988), 534-541.

Fundamental issues of technology transfer, training, problem selection, staffing, corporate politics, and more, are explored.

Curtis88

Curtis, Bill, Herb Krasner, and Neil Iscoe. "A Field Study of the Software Design Process for Large Systems." *Comm. ACM 31*, 11 (Nov. 1988), 1268-1287.

A rare study of how software development *really* takes place. The authors interviewed personnel from 17 large software projects. Using a layered behavioral model, they analyze the impact on quality and productivity of shifting requirements, inadequate domain knowledge, and communication problems.

Cyert87

Cyert, R. M. *Bayesian Analysis and Uncertainty in Economic Theory*. Totowa, N.J.: Rowman & Littlefield, 1987.

Looks at statistical methods used in economics where data are incomplete.

Dart87

Dart, Susan A., Robert J. Ellison, Peter H. Feiler, and Nico Habermann. "Software Development Environments." *Computer 20*, 11 (Nov. 1987), 18-28.

This taxonomy of software development environments serves as useful reading for discussions of resource acquisition, allocation, and training.

DeMarco82

DeMarco, Tom. *Controlling Software Projects*. New York: Yourdon Press, 1982.

This book concentrates heavily on metrics and the examination of quality factors as the basis for management of software projects. Essentially, DeMarco makes the case that you cannot control what you cannot measure. Useful for finding ways to integrate metrics into an organization.

DeMarco87

DeMarco, Tom and Timothy Lister. *Peopleware: Productive Projects and Teams*. New York: Dorset House, 1987.

A collection of essays on managing projects and the people who carry them out by a pair of authors with substantial project management and consulting experience. The essays challenge conventional management wisdom and emphasize how creating the right work environment can increase productivity.

Deming86

Deming, W. Edwards. *Out of the Crisis*. Cambridge, Mass.: MIT Center for Advanced Engineering Study, 1986.

Deming advocates the transformation of American management to focus on increasing quality and productivity within organizations. He offers 14 guiding principles for achieving this transformation, and he identifies the "deadly diseases" and obstacles that currently are barriers to doing so. Deming's principles provided the framework for the development of postwar Japanese industry, and he is widely credited with guiding its tremendous success.

DoD88
DoD. *Military Standard for Defense System Software Development.* DOD-STD-2167A, U. S. Department of Defense, Washington, D.C., 29 February 1988.

Due to its completeness and maturity as the successor to military standards of the preceding two decades, this is one of the best available examples to use when discussing standards.

EPOS88
EPOS. SPS, Inc., New York, 1988.

A comprehensive software engineering environment for workstations that includes project management tools.

Evans83
Evans, M. W., P. H. Piazza, and J. B. Dolkas. *Principles of Productive Software Management*. New York: John Wiley, 1983.

This book describes a methodology for software management and the associated control techniques for the entire development process, as practiced by the authors at Ford Aerospace and Communications Corporation and several other companies.

Fagan76
Fagan, M. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems J. 15*, 3 (1976), 182-211.

A "must read" classic paper that introduces the whole concept of software inspections. Sample forms, checklists, and experimental data from IBM are also presented.

Fairley85
Fairley, Richard E. *Software Engineering Concepts*. New York: McGraw-Hill, 1985.

Of the plethora of introductory texts on software engineering, Fairley's has, arguably, the best set of references. His text is compact, living up to the title in that only "concepts" are presented; but extensive references are given for each topic. The first three chapters are most useful in the management area.

Fairley88
Fairley, Richard E. "A Guide to Preparing Software Project Management Plans." In *Tutorial: Software Engineering Project Management*, Richard H. Thayer, ed. Washington, D.C.: IEEE Computer Society Press, 1988, 257-264.

This is written in the spirit of the various IEEE standards for plans (quality assurance, configuration management, etc.). One of the best parts of this guide is the conceptual introduction to the software development process and the relation of the project plan to it.

Feller68
Feller, W. *An Introduction to Probability Theory and Its Applications, 3rd Ed*. New York: John Wiley, 1967.

This book is used to teach probability and statistics using a mathematical approach.

Ferrari78
Ferrari, D. *Computer Systems Performance Evaluation*. Englewood Cliffs, N.J.: Prentice-Hall, 1978.

This book looks at performance evaluation using measurement, simulation, and analytic techniques. It then applies these to solve problems characteristic of methodology selection, design alternatives, and product improvement.

Freeman87
Freeman, Peter. *Software Perspectives: The System is the Message*. Reading, Mass.: Addison-Wesley, 1987.

This book is too general to be a single text for software project management courses. However, it does emphasize the importance of environments and preparing for transition of a product.

Friedman87
Friedman, Marc S., and Mary J. Hildebrand. "Computer Litigation: A Buyer's Theories of Liability." *The Computer Lawyer 4*, 12 (Dec. 1987), 34-38.

This article provides useful material for a discussion of software users' rights and responsibilities.

Gilb88
Gilb, Tom. *Principles of Software Engineering Management*. Reading, Mass.: Addison-Wesley, 1988.

Gilb has written a nearly complete discussion of project management from a unique viewpoint.

Glass81
Glass, Robert L., and Ronald A. Noiseux. *Software Maintenance Guidebook*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

This book is useful for background on managing maintenance.

Gourd82

Gourd, Roger S. "Self-Assessment Procedure X: A Self-Assessment Procedure Dealing with Software Project Management." *Comm. ACM 25*, 12 (Dec. 1982), 883-887.

Self-assessments are a familiar but irregular feature of the *Communications of the ACM*. This one is weaker than some, being most useful for beginners to the subject, helping to identify which buzzword categories are most in need of development.

Harvey86

Harvey, Katherine E. *Summary of the SEI Workshop on Software Configuration Management.* CMU/SEI-86-TR-5, Software Engineering Institute, Pittsburgh, Pa., Dec. 1986.

This is good reference material for presenting basic concepts on configuration management.

Holmes82

Holmes, Robert A. "Application of Article Two of the Uniform Commercial Code to Computer System Acquisitions." *Rutgers Computer and Technology Law J. 9*, 1 (1982), 1-26.

This article address such issues as product misrepresentation, purchasing of systems, warranties, and unconscionability.

Howes84

Howes, Norman R. "Managing Software Development Projects for Maximum Productivity." *IEEE Trans. Software Eng. SE-10*, 1 (Jan. 1984), 27-35.

Abstract: *In the area of software development, data processing management often focuses more on coding techniques and system architecture than on how to manage the development. In recent years, "structured programming" and "structured analysis" have received more attention than the techniques software managers employ to manage. Moreover, these coding and architectural considerations are often advanced as the key to a smooth running, well managed project.*

This paper documents a philosophy for software development and the tools used to support it. Those management techniques deal with quantifying such abstract terms as "productivity," "performance," and "progress," and with measuring these quantities and applying management controls to maximize them. The paper also documents the application of these techniques on a major software development effort.

Howes suggests a method for "packing" work, and details the components involved in planning, executing, and evaluating the software development process.

Humphrey87

Humphrey, Watts S. *Managing for Innovation: Leading Technical People*. Englewood Cliffs, N.J.: Prentice-Hall, 1987.

If there is a *Mythical Man-Month* for managers, this is probably it. Humphrey has collected his and his IBM colleagues' collective experiences in leading technical individuals and teams into this compact, readable, and immediately useful book. It is best to read a chapter at a time, with some reflection between segments—there is just too much in a typical chapter to absorb it adequately in conjunction with its neighbors.

Humphrey88

Humphrey, Watts S. "Characterizing the Software Process: A Maturity Framework." *IEEE Software 5*, 3 (March 1988), 73-79.

Humphrey presents a framework for the evolution of the software development process.

Humphrey89

Humphrey, Watts S. *Managing the Software Process*. Reading, Mass.: Addison-Wesley, 1989.

A genuine handbook for managers. It is very detailed and complete.

IBM85

Issue on Quality and Productivity. G. M. Gershon, ed. *IBM Systems J. 24*, 2 (1985).

This issue is devoted to articles concerned with software *quality* and *productivity*. It concentrates primarily on the development of large systems and contains measurements of organizations that "produce several million lines of new and changed code every year." It contains articles on programming process, automation of the development process, software methodologies, defect prevention approaches, software engineering education, and productivity measurements. It contains a lot of metrics and analyses of the effectiveness of different approaches.

IEEE83

IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. New York: IEEE, 1983. ANSI/IEEE Std 729-1983.

IEEE84

IEEE. *IEEE Standard for Software Quality Assurance Plans*. New York: IEEE, 1984. ANSI/IEEE Std 730-1984.

IEEE87
IEEE. *IEEE Standard for Software Project Management Plans*. New York: IEEE, 1987. IEEE Std 1058.1-1987.

Jackson82
Jackson, J. H., and C. P. Morgan. *Organization Theory: A Macro Perspective for Management 2nd Ed*. Englewood Cliffs, N.J.: Prentice-Hall, 1982.

A useful book on organization theories.

Jullig86
Jullig, Richard, *et al*. *Knowledge-Based Software Project Management.* Technical Report KES.U.-87.3, Kestrel Institute, Palo Alto, Calif., Nov. 1986.

This is a description of the knowledge-based Project Management Assistant, developed as an extension to the Refine environment. Recent research into new models of time (time as intervals instead of absolute elapsed time) are incorporated into the system. Several interesting concepts are presented, but the bulk of the material is beyond-leading-edge, thus difficult to apply with present tools.

Kemerer87
Kemerer, Chris F. "An Empirical Validation of Software Cost Estimation Models." *Comm. ACM 30*, 5 (May 1987), 416-429.

Abstract: *Practitioners have expressed concern over their inability to accurately estimate costs associated with software development. This concern has become even more pressing as costs associated with development continue to increase. As a result, considerable research attention is now directed at gaining a better understanding of the software-development process as well as constructing and evaluating software cost estimating tools. This paper evaluates four of the most popular algorithmic models used to estimate software costs (SLIM, COCOMO, Function Points, and ESTIMACS). Data on 15 large completed business data-processing projects were collected and used to test the accuracy of the models'* ex post *effort estimation. One important result was that Albrecht's Function Points effort estimation model was validated by the independent data provided in this study [3]. The models not developed in business data-processing environments showed significant need for calibration. As models of the software-development process, all of the models tested failed to sufficiently reflect the underlying factors affecting productivity. Further research will be required to develop understanding in this area.*

This article is useful to have students read after presenting cost estimation models based on source-lines-of-code. It demonstrates the limitations of the currently available models and introduces a different method, function points, which can then be discussed using [Albrecht83].

Kepner65
Kepner, C. H., and B. B. Tregoe. *The Rational Manager*. New York: McGraw-Hill, 1965.

A systematic approach to problem solving and decision making. Stresses techniques useful in determining problem causes.

Kerin87
Kerin, R. A., and R. A. Peterson. *Strategic Marketing Problems: Cases and Comments, 4th Ed*. Boston, Mass.: Allyn & Bacon, 1987.

This text contains decision making and management case studies as they apply to marketing.

Kotler88
Kotler, P. *Marketing Planning: Analysis, Planning, Implementation, and Control, 6th Ed*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

A popular text on marketing.

Lavenberg83
Lavenberg, S. S. *Computer Performance Modeling Handbook*. New York: Academic Press, 1983.

This book is a collection of papers, most by Lavenberg, covering a number of modeling approaches including analysis, simulation, and validation of computer performance models. This is considered by some to be the reference manual for modeling practitioners who concentrate on hardware modeling.

MacProject89
MacProject II. Claris Corp., Mountain View, Calif., 1989.

A substantially enhanced version of MacProject, project planning software for the Apple Macintosh. MacProject II provides a variety of tools for planning projects and tracking their progress, including PERT and Gantt charts, resource histograms, and calendars. It features color displays, allows multiuser data access, and adjustable-size output.

Mann88
Mann, Nancy R. "Why it Happened in Japan and Not in the U.S." *Chance: New Directions for Statistics and Computing 1*, 3 (Summer 1988), 8-15.

The story of how W. Edwards Deming's statistical quality-control methodology came to be embraced

in Japan, after it failed to take hold in the U.S. The crucial factor, Mann asserts, was the buy-in of Japanese management. The competitive advantage this gave Japanese industry should not be lost on American software developers.

Mantei81

Mantei, M. "The Effect of Programming Team Structures on Programming Tasks." *Comm. ACM 24*, 3 (March 1981), 106-113.

Summary: *The literature recognizes two group structures for managing programming projects: Baker's chief programmer team and Weinberg's egoless team. Although each structure's success in project management can be demonstrated, this success is clearly dependent on the type of programming task undertaken. Here, for the purposes of comparison, a third project organization which lies between the other two in its communication patterns and dissemination of decision-making authority is presented. Recommendations are given for selecting one of the three team organizations depending on the task to be performed.*

This paper discusses the effectiveness of various organization styles in software development. It is included in [Thayer88].

McGill84

McGill, James P. "The Software Engineering Shortage: A Third Choice." *IEEE Trans. Software Eng. SE-10*, 1 (Jan. 1984), 42-48.

Abstract: *As interest in the concepts and methods of software engineering increases, many companies, particularly in aerospace, find it difficult to acquire software developers with the desired skills. The option of full-time, company-based training is discussed with suggestions for implementation. Lessons learned from the actual implementation of such a program are discussed along with possible directions for future evolution.*

This article addresses the software crisis, as well as what happens when companies actually get involved with internal training programs.

Metzger81

Metzger, Phillip W. *Managing a Programming Project, 2nd Ed*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

Yet another of the ex-IBM managers that grew up with the software industry (Brooks and Humphrey are two others) puts pen to paper. Metzger has a very engaging, informal style. Part I of this book is a travelogue through the traditional phases of the waterfall life-cycle model, with instructions as to the role of the manager in each phase. Part II contains an annotated outline of the key documents produced at each step. This book serves as a good introduction to the process of software engineering in general. However, it is quite spare and should be used in conjunction with [Metzger87].

Metzger87

Metzger, Phillip W. *Managing Programming People: A Personal View*. Englewood Cliffs, N.J.: Prentice-Hall, 1987.

Metzger concentrates on the most important aspect of a software project—the people involved in making the product. He devotes a chapter to each of the key types of personnel: analyst, designer, programmer, tester, support staff, and also the customer. There is a wealth of experience contained in compact spaces, and the art chosen to illustrate key points outdoes that of *The Mythical Man-Month*. The sections on the specific people can be matched with life cycle phases in [Metzger81].

Mills83

Mills, Harlan D. *Software Productivity*. Boston, Mass.: Little, Brown, 1983. Reprinted by Dorset House in 1988.

A collection of Mills's papers from the late 1960s to the early 1970s. It is possible to trace his thinking on programming team organization.

NASA86a

NASA. *Software Management Plan Data Item Description.* NASA-Sfw-DID-02-ADA, National Aeronautics and Space Administration, Office of Safety, Reliability, Maintainability, and Quality Assurance, Washington D.C., 1986.

This DID outlines what should be in a project plan and serves as reference material for the discussion on project plan contents.

NASA86b

NASA. *Lessons-Learned Document Data Item Description.* NASA-Sfw-DID-41, National Aeronautics and Space Administration, Office of Safety, Reliability, Maintainability, and Quality Assurance, Washington D.C., 1986.

The Lessons-Learned Document (LLD) describes the management and/or development of a project and the experience gained from the execution of the project, assesses the project's strengths and weaknesses, and evaluates the results.

Newell86

Newell, Allen. "Response: The Models Are Broken, the Models Are Broken!" *U. of Pittsburgh Law Review 47*, 4 (Summer 1986), 1023-1035.

This is insightful reading for discussions on patentability and copyright.

Pressman87

Pressman, Roger S. *Software Engineering: A Practitioner's Approach, 2nd Ed*. New York: McGraw-Hill, 1987.

Good introductory text for software engineering courses. It contains much material on management.

Pressman88

Pressman, Roger S. *Making Software Engineering Happen: A Guide for Instituting the Technology*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

The author of a popular software engineering textbook here addresses the problem of how to introduce software engineering techniques and tools into the workplace. In this guidebook for managers, Pressman introduces and discusses the "software engineering implementation life cycle."

Putnam78

Putnam, Lawrence H. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem." *IEEE Trans. Software Eng. SE-4*, 4 (July 1978), 345-361.

Abstract: *Application software development has been an area of organizational effort that has not been amenable to the normal managerial and cost controls. Instances of actual costs of several times the initial budgeted cost , and a time to initial operational capability sometimes twice as long as planned are more often the case than not.*

A macromethodology to support management needs has now been developed that will produce accurate estimates of manpower, costs, and times to reach critical milestones of software projects. There are four parameters in the basic system and these are in terms managers are comfortable working with—effort, development time, elapsed time, and a state-of-technology parameter.

The system provides managers sufficient information to assess the financial risk and investment value of a new software development project before it is undertaken and provides techniques to update estimates from the actual data stream once the project is underway. Using the technique developed in the paper, adequate analysis for decisions can be made in an hour or two using only a few quick reference tables and a scientific pocket calculator.

This article should be included in readings for students about estimation techniques and models.

Putnam79a

Putnam, Lawrence H. and Ann Fitzsimmons. "Estimating Software Costs, Part I." *Datamation 25*, 10 (Sept. 1979), 188-190, 194, 198.

This and the following two articles present a composite costing model that considers more factors than the lower-level COCOMO models. There is no evaluation of the techniques presented, but the authors appear to have considerable practical savvy.

Putnam79b

Putnam, Lawrence H. and Ann Fitzsimmons. "Estimating Software Costs, Part II." *Datamation 25*, 11 (Oct. 1979), 171-172, 176, 178.

Putnam79c

Putnam, Lawrence H. and Ann Fitzsimmons. "Estimating Software Costs, Part III." *Datamation 25*, 12 (Nov. 1979), 137-138, 140.

Putnam80

Putnam, L. H., ed. *Tutorial: Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers*. New York: IEEE Computer Society Press, 1980.

A collection of papers given at COMSAC80, October 27-31, 1980. Many of the papers are by Putnam. His approach to estimating has since been referred to as the Putnam model of cost estimation.

Radice85

Radice, R. A., *et al*. "A Programming Process Architecture." *IBM Systems J. 24*, 2 (1985), 79-90.

Abstract: *The Programming Process Architecture is a framework describing required activities for an operational process that can be used to develop system or application software. The architecture includes process management tasks, mechanisms for analysis and development of the process, and product quality reviews during the various stages of the development cycle. It requires explicit entry criteria, validation, and exit criteria for each task in the process, which combined form the "essence" of the architecture. The architecture describes requirements for a process needing no new invention, but rather using the best proven methodologies, techniques, and tools available today. This paper describes the Programming Process Architecture and its use, emphasizing the reasons for its development.*

This article (and several other articles in the same issue of this journal) describes an "architecture" for the automation of the software development process. Although, the automation of the architecture may not have progressed as described here, this arti-

cle is representative of the type of thinking and research that was going on at that time in academia and industry.

Radice88

Radice, R. A., and R. W. Philips. *Software Engineering*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

An industrial approach to software engineering.

Sauer81

Sauer, C. H. and Mani K. Chandy. *Computer Systems Performance Modeling*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

This book is an interesting treatise on performance modeling. It covers general principles, Markovian and other queuing models, approximation techniques, simulation, measurement, and parameter estimation. It contains six modeling case studies and discusses the management aspects of modeling projects.

Scacchi84

Scacchi, Walt. "Managing Software Engineering Projects: A Social Analysis." *IEEE Trans. Software Eng. SE-10*, 1 (Jan. 1984), 49-58.

Abstract: *Managing software engineering projects requires an ability to comprehend and balance the technological, economic, and social bases through which large software systems are developed. It requires people who can formulate strategies for developing systems in the presence of ill-defined requirements, new computing technologies, and recurring dilemmas with existing computing arrangements. This necessarily assumes skill in acquiring adequate computing resources, controlling projects, coordinating development schedules, and employing and directing competent staff. It also requires people who can organize the process for developing and evolving software products with locally available resources. Managing software engineering projects is as much a job of social interaction as it is one of technical direction. This paper examines the social arrangements that a software manager must deal with in developing and using new computing systems, evaluating the appropriateness of software engineering tools or techniques, directing the evolution of a system through its life cycle, organizing and staffing software engineering projects, and assessing the distributed costs and benefits of local software engineering practices. The purpose is to underscore the role of social analysis of software engineering practices as a cornerstone in understanding what it takes to productively manage software projects.*

This article is appropriate reading for a discussion of people management.

Scott76

Scott, W. G., and T. R. Mitchell. *Organization Theory: A Structural and Behavioral Analysis, 3rd Ed.* Homewood, Ill.: Irwin, 1976.

Classic approach in organization theory.

Shannon75

Shannon, R. E. *Systems Simulation: The Art and Science*. Englewood Cliffs, N.J.: Prentice-Hall, 1975.

This book covers the life cycle of system simulation. It goes from systems investigation through validation and analysis. It covers management aspects, model translation, planning and design of experiments. It includes six case studies in simulation.

Shere88

Shere, K. D. *Software Engineering and Management*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

This book is intended for the computer professional who needs to gain a system-level perspective of software development. It contains seven chapters on the system development life cycle, including discussions of risk management and cost estimation. It uses a case study to discuss structured design and database design and then addresses such subjects as quality assurance, capacity planning, and reliability. It concludes with a "case study of a systems engineer and integration job."

Spicer83

Spicer, Joseph C. *A Spiral Approach to Software Engineering Project Management Education.* Technical Report TR-83-05, School of Information Technology, Wang Institute of Graduate Studies, Tyngsboro, Mass., Dec. 1983.

Abstract: *This paper describes the experiences of an instructor, experienced in management, teaching a software engineering project management course at a school that specializes in software education. A "spiral approach" was used to provide for the parallel acquisition of management knowledge and experience, while building on recently acquired skills in software technology and developing confidence through the successful development of a software product. This paper describes the reason for selecting the approach, the course content, the observation of the students and the instructor, the advantages and disadvantages of the approach as applied, and conclusions. Course objectives were met and course evaluations by students indicated a much higher level of acceptance than previous traditional approaches. The observations indicated that the approach may extend beyond the classroom*

to the industrial setting where on the job training, career path planning, and management development are of concern.

This report reflects on a single attempt to adapt the spiral approach to project management education. The course proceeded through individual, small, and medium sized projects, adding techniques and knowledge as the scope expanded. This approach contrasts with the organization of the available project management texts, which are still quite linear. This approach has its merits and should be considered in the development of project management courses.

Stuckenbruck81

Stuckenbruck, Linn C. "The Matrix Organization." In *A Decade of Project Management: Selected Readings from the* Project Management Quarterly, *1970 through 1980*, John R. Adams and Nicki S. Kirchof, eds. Drexel Hill, Pa.: Project Management Institute, 1981, 157-169.

This paper describes the matrix form of organizing a development organization. In general, it can be thought that the staff are organized by expertise (architects, designers, coders, testers, computer support, etc.) into manageable groups along the horizontal axis. The actual development projects are along the vertical axis. They get commitments from the managers on the vertical axis for the people to do their projects.

Symons88

Symons, Charles R. "Function Point Analysis: Difficulties and Improvements." *IEEE Trans. Software Eng. SE-14*, 1 (Jan. 1988), 2-11.

Abstract: *The method of Function Point Analysis was developed by Allan Albrect to help measure the size of a computerized business information system. Such sizes are needed as a component of the measurement of productivity in system development and maintenance activities, and as a component of estimating the effort needed for such activities. Close examination of the method shows certain weaknesses, and the author proposes a partial alternative. The paper describes the principles of this "Mark II" approach, the results of some measurements of actual systems to calibrate the Mark II approach, and conclusions on the validity and applicability of function point analysis generally.*

This article is excellent for the presentation and contrast of the two function point methods (Albrecht's and Symons's).

Tausworthe80

Tausworthe, Robert C. "The Work Breakdown Structure in Software Project Management." *J. Syst. and Software 1*, 3 (August 1980), 181-186.

Abstract: *The work breakdown structure (WBS) is a vehicle for breaking an engineering project down into subproject, tasks, subtasks, work packages, and so on. It is an important planning tool which links objectives with resources and activities in a logical framework. It becomes an important status monitor during the actual implementation as the completions of subtasks are measured against the project plan. Whereas the WBS has been widely used in many other engineering applications, it has seemingly only rarely been formally applied to software projects, for various reasons. Recent successes with software project WBSs, however, have clearly indicated that the technique can be applied and have shown the benefits of such a tool in management of these projects.*

This paper advocates and summarizes the use of the WBS in software implementation projects. It also identifies some of the problems people have had generating software WBSs, and the need for standard checklists of items to be included.

This paper presents the work breakdown structure as a viable planning tool for software project managers and includes an outline of the detailed work breakdown structure.

Thayer80

Thayer, Richard H., Arthur Pyster, and Roger C. Wood. "The Challenge of Software Engineering Project Management." *Computer 13*, 8 (Aug. 1980), 51-58.

This article is a reasonably well-conducted survey of project managers, programmers, engineers, and students to determine the perceived biggest problems in software project management and potential solutions. Due to deficiencies in the instrument (not enough room left for describing known solutions), the problems certainly get more play than the answers. The data seem still to be valid.

Thayer88

Thayer, R. H., ed. *Tutorial: Software Engineering Project Management*. Washington, D.C.: IEEE Computer Society Press, 1988.

This tutorial contains many of the important papers relevant to software engineering and project management. Included are papers on software engineering, project management, planning, organizing, staffing, directing, and controlling a software engineering project.

Walpole85
Walpole, R. E., and R. H. Myers. *Probability and Statistics for Engineers and Scientists, 3rd Ed*. New York: MacMillan, 1985.

The mathematical approach to probability and statistics.

Weinberg71
Weinberg, G. M. *The Psychology of Computer Programming*. New York: Van Nostrand, 1971.

This book is a classic that looks at the human element of computer programming. It investigates in detail the behavior and thought processes of computer programmers at the time.

Weinberg84
Weinberg, Gerald M., and Daniel P. Freedman. "Reviews, Walkthroughs, and Inspections." *IEEE Trans. Software Eng. SE-10*, 1 (Jan. 1984), 68-72.

Abstract: *Formal technical reviews supply the quality measurement to the "cost effectiveness" equation in a project management system. There are several unique formal technical review procedures, each applicable to particular types of technical material and to the particular mix of the Review Committee. All formal technical reviews produce reports on the overall quality for project management, and specific technical information for the producers. These reports also serve as an historic account of the systems development process. Historic origins and future trends of formal and informal technical reviews are discussed.*

This article fits well in the discussion of tracking, reviewing, and adjusting goals.

Weinberg86
Weinberg, Gerald M. *Becoming a Technical Leader*. New York: Dorset House, 1988.

Weinberg presents his "MOI" model of technical leadership in this "how-to" book. The successful problem-solving leader, he asserts, has strong skills in three areas: *m*otivation, *o*rganization, and *i*nnovation.

Weisbord88
Weisbord, Marvin R. *Productive Workplaces: Organizing and Managing for Dignity, Meaning, and Community*. San Francisco: Jossey-Bass, 1988.

Weisbord reviews the significant movements in management science and offers his own view of how to design and manage more productive workplaces that meet more successfully the needs of both organizations and employees. The particular significance of this approach for software managers is that it recognizes the rapid changes that occur in the modern workplace and incorporates this reality into its management guidelines.

Curriculum Modules and Related Publications

Berztiss88
Bertziss, Alfs T., and Mark A. Ardis. *Formal Verification of Programs.* Curriculum Module SEI-CM-20-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Dec. 1988.

Capsule Description: *This module introduces formal verification of programs. It deals primarily with proofs of sequential programs, but also with consistency proofs for data types and deduction of particular behaviors of programs from their specifications. Two approaches are considered: verification after implementation that a program is consistent with its specification, and parallel development of a program and its specification. An assessment of formal verification is provided.*

Brackett88
Brackett, John W. *Software Requirements.* Curriculum Module SEI-CM-19-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Dec. 1988.

Capsule Description: *This curriculum module is concerned with the definition of software requirements—the software engineering process of determining what is to be produced—and the products generated in that definition. The process involves:*

- *requirements identification,*
- *requirements analysis,*
- *requirements representation,*
- *requirements communication, and*
- *development of acceptance criteria and procedures.*

The outcome of requirements definition is a precursor of software design.

Brown87
Brown, Brad. *Assurance of Software Quality.* Curriculum Module SEI-CM-7-1.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., July 1987.

Capsule Description: *This module presents the underlying philosophy and associated principles and practices related to the assurance of software qual-*

ity. It includes a description of the assurance activities associated with the phases of the software development life-cycle (e.g., requirements, design, test, etc.).

Budgen89

Budgen, David. *Introduction to Software Design.* Curriculum Module SEI-CM-2-2.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Jan. 1989.

Capsule Description: *This curriculum module provides an introduction to the principles and concepts relevant to the design of large programs and systems. It examines the role and context of the design activity as a form of problem-solving process, describes how this is supported by current design methods, and considers the strategies, strengths, limitations, and main domains of application of these methods.*

Collofello88a

Collofello, James S. *Introduction to Software Verification and Validation.* Curriculum Module SEI-CM-13-1.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Dec. 1988.

Capsule Description: *Software verification and validation techniques are introduced and their applicability discussed. Approaches to integrating these techniques into comprehensive verification and validation plans are also addressed. This curriculum module provides an overview needed to understand in-depth curriculum modules in the verification and validation area.*

Collofello88b

Collofello, James S. *The Software Technical Review Process.* Curriculum Module SEI-CM-3-1.5, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., June 1988.

Capsule Description: *This module consists of a comprehensive examination of the technical review process in the software development and maintenance life cycle. Formal review methodologies are analyzed in detail from the perspective of the review participants, project management and software quality assurance. Sample review agendas are also presented for common types of reviews. The objective of the module is to provide the student with the information necessary to plan and execute highly efficient and cost effective technical reviews.*

Cross88

Cross, John, ed. *Support Materials for The Software Technical Review Process.* Support Materials SEI-SM-3-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1988.

Glass88

Glass, Robert L. *An Overview of Technical Communication for the Software Engineer.* Curriculum Module SEI-CM-18-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1988.

Capsule Description: *This module presents the fundamentals of technical communication that might be most useful to the software engineer. It discusses both written and oral communication.*

Although the information in this module is general, the bibliography contains helpful references.

Mills88

Mills, Everald E. *Software Metrics.* Curriculum Module SEI-CM-12-1.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Dec. 1988.

Capsule Description: *Effective management of any process requires quantification, measurement, and modeling. Software metrics provide a quantitative basis for the development and validation of models of the software development process. Metrics can be used to improve software productivity and quality. This module introduces the most commonly used software metrics and reviews their use in constructing models of the software development process. Although current metrics and models are certainly inadequate, a number of organizations are achieving promising results through their use. Results should improve further as we gain additional experience with various metrics and models.*

Morell89

Morell, Larry J. *Unit Testing and Analysis.* Curriculum Module SEI-CM-9-1.2, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1989.

Capsule Description: *This module examines the techniques, assessment, and management of unit testing and analysis. Testing and analysis strategies are categorized according to whether their coverage goal is functional, structural, error-oriented, or a combination of these. Mastery of the material in this module allows the software engineer to define, conduct, and evaluate unit tests and analyses and to assess new techniques proposed in the literature.*

Perlman88

Perlman, Gary. *User Interface Development.* Curriculum Module SEI-CM-17-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1988.

Capsule Description: *This module covers the issues, information sources, and methods used in the design, implementation, and evaluation of* user interfaces, *the parts of software systems designed to interact with people. User interface* design *draws on the experiences of designers, current trends in input/output technology, cognitive psychology, human factors (ergonomics) research, guidelines and standards, and on the feedback from evaluating working systems. User interface* implementation *applies modern software development techniques to building user interfaces. User interface* evaluation *can be based on empirical evaluation of working systems or on the predictive evaluation of system design specifications.*

Samuelson88

Samuelson, Pamela, and Anne Crawford Martin. *Software Development and Licensing Contracts.* Curriculum Module SEI-CM-15-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1988.

Capsule Description: *This module provides an overview of contract law and principles that form the framework within which software development contracts and software licensing contracts are made and enforced. It discusses some of the substantive provisions of software contracts and licenses, along with the "default setting" that intellectual property law provides when parties to a contract have failed to make their rights and responsibilities explicit. It also discusses limitations that laws place on the freedom of developers and users to make certain kinds of contractual arrangements for software.*

Samuelson89

Samuelson, Pamela, and Kevin Deasy. *Intellectual Property Protection For Software.* Curriculum Module SEI-CM-14-2.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., July 1989.

Capsule Description: *This module provides an overview of the U.S. intellectual property laws that form the framework within which legal rights in software are created, allocated, and enforced. The primary forms of intellectual property protection that are likely to apply to software are copyright, patent, and trade secret laws, which are discussed with particular emphasis on the controversial issues arising in their application to software. Also included is a brief introduction to government software acquisition regulations, trademark, trade dress, and related unfair competition issues that may affect software engineering decisions, and to the Semiconductor Chip Protection Act.*

Scacchi87

Scacchi, Walt. *Models of Software Evolution: Life Cycle and Process.* Curriculum Module SEI-CM-10-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Oct. 1987.

Capsule Description: *This module presents an introduction to models of software system evolution and their role in structuring software development. It includes a review of traditional software life-cycle models as well as software process models that have been recently proposed. It identifies three kinds of alternative models of software evolution that focus attention to either the products, production processes, or production settings as the major source of influence. It examines how different software engineering tools and techniques can support life-cycle or process approaches. It also identifies techniques for evaluating the practical utility of a given model of software evolution for development projects in different kinds of organizational settings.*

Tomayko87a

Tomayko, James E. *Software Configuration Management.* Curriculum Module SEI-CM-4-1.3, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., 1987.

Capsule Description: *Software configuration management encompasses the disciplines and techniques of initiating, evaluating, and controlling change to software products during and after the development process. It emphasizes the importance of configuration control in managing software production.*

Tomayko87b

Tomayko, James E. *Teaching a Project-Intensive Introduction to Software Engineering.* Technical Report SEI-87-TR-20, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1987.

This document contains a syllabus and description of an introduction to software engineering course that demonstrates how to integrate management material into such a course.