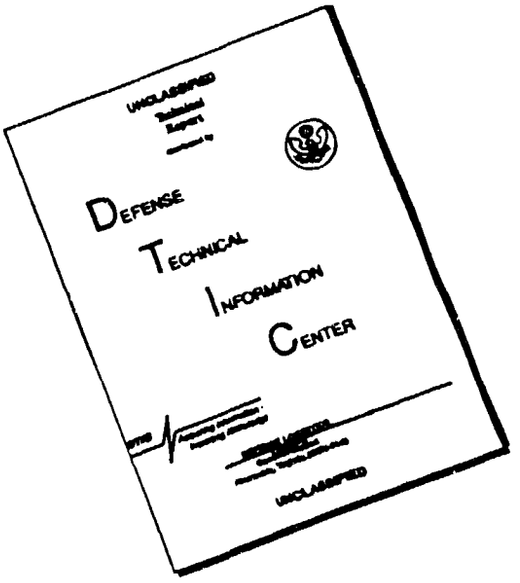
Carnegie-Mellon University
Software Engineering Institute

Support Materials for

The Software Technical Review Process

Support Materials SEI-SM-3-1.0

DISCLAIMER NOTICE



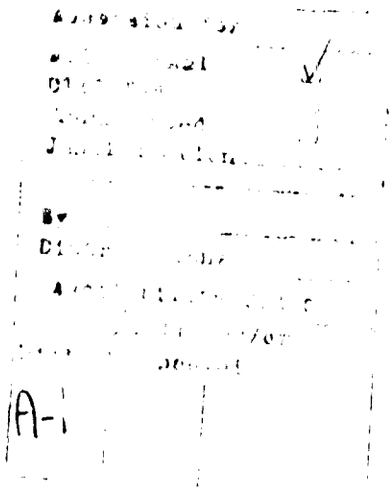
THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Support Materials
for

The Software Technical Review Process

SEI Support Materials SEI-SM-3-1.0

April 1988



Edited by

John A. Cross
Indiana University of Pennsylvania

Carnegie Mellon University
Software Engineering Institute

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

Draft For Public Review

This technical report was prepared for the

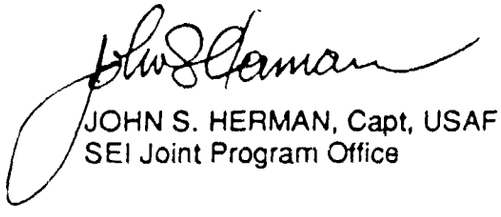
SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER



JOHN S. HERMAN, Capt, USAF
SEI Joint Program Office

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

Contents

The Software Technical Review Process

Support Materials Revision History

Version 1.0 (April 1988) Draft for public review

Introduction

The purpose of these support materials is to facilitate the teaching and use of software technical reviews in university courses on software engineering subjects. They are intended to supplement the SEI curriculum module, *The Software Technical Review Process* (CM-3)[1]. Although it has been developed for use in a university setting, much of the material is appropriate for the training of professionals as well.

The user of this material should consider the variety of ways in which technical review processes may be incorporated into software-related curricula. For example:

- Instruction in review concepts and procedures gives students the necessary foundation for participating in or implementing a technical review process.
- Software technical reviews may also be used in the teaching of general concepts of software engineering by requiring students to inspect and discuss concrete examples of software development artifacts.
- Practice in conducting reviews provides students with an essential means of gaining competence as reviewers. Students may be required to conduct reviews within the context of software development projects.
- Group reviews give students an opportunity to gain experience and knowledge that will help them improve their effectiveness as team members. Group reviews may be used to provide experience with group interaction, or they may be used as a means of studying group dynamics.

The materials included here are particularly designed to support an inspection methodology, or work product review, as it is described in Section 6 of CM-3. These materials are expected to meet the critical needs for support materials in this area. Practical knowledge of the inspection methodologies of [Fagan76] and [Freedman82], together with general knowledge of software development artifacts, can provide the basis for using methodologies such as walk-through [Yourdon85], audit [IEEE85], in-process software analysis [Howden82], and "work reading" [Ledgard86].

Although lecture notes are included, the emphasis is on actual practice by groups of students. The explanatory text defines basic issues that an instructor should address when structuring a learning activity that includes a software technical review process. Rationale for alternative actions is also given.

The instructor should consider the concerns listed in the Instructor's Checklist (p. 22) in preparing for a software technical review. Some support materials may be used as is, but others will have to be modified to suit the needs of a particular context. The introductory text provided with each item attempts to put the item in an appropriate educational context and points to specific modifications that might be required.

Part I is designed to support direct teaching of fundamental concepts. The part includes notes for lectures or student handouts, sample software defects, and questions that can be used for knowledge assessment. Also included are guidelines for teaching concepts using practice reviews.

---

[1]Collofello, James S. *The Software Technical Review Process*. Curriculum Module SEI-CM-3-1.3, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1988.

Part II, *Implementing a Technical Software Review as a Learning Activity*, includes a variety of materials to be used in preparing for practice reviews primarily designed to teach about software development. These materials are accompanied by sample software artifacts to be used for practice exercises.

Part III contains materials that may be used to facilitate technical reviews of in-process software artifacts, particularly in the context of a group project. Note that the materials in **Part III** may also be useful in practice activities such as those in **Part II**.

Part I: Teaching Software Reviewing Concepts

Notes on Software Technical Reviewing

John A.Cross
Indiana University of Pennsylvania

Description: General notes on the software technical review process and the software development life cycle. These notes were used as handouts in an undergraduate course, *Software Engineering Concepts*, Computer Science 319, during three different semesters at Indiana University of Pennsylvania.

Purpose: To provide students with general background material not readily available in textbooks.

Procedure: These notes can be used as a starting point for classroom lectures or they can be given to students as handouts. (Instructors should be aware that the notes include several specific references to *Software Engineering Concepts*.) Additional sources of material appear in the bibliography of CM-3, which is reproduced beginning on p. 73.

Commentary: Although there are many other sources of information on software development life cycle concepts, the following notes are included because they contain concepts that are fundamental to software technical reviews in a concise form. Deutsch and Willis [Deutsch88] provide additional discussion of concepts of software technical reviews and software quality.

Notes on the Software Technical Review Process

Software reviewing is a general term applied to techniques for the use of human intellectual power to detect flaws in software during the process of software development. Because software reviewing does not necessarily involve any automatic system execution, it is frequently categorized as a "static testing" approach to software quality assurance. The most notable feature of popular techniques for software reviewing is the use of "egoless" group interaction. The term egoless generally connotes that group members focus on the task of understanding a piece of software and noting ways to improve it, rather than on interpersonal factors or the failure of an individual to deal with possible defects.

An IEEE draft standard for software reviews and audits defines the basic concepts and terminology of software reviewing [IEEE85]. For example, a walk-through (see [Yourdon85]) is a dynamic presentation of the behavior of the software element in various scenarios. An inspection (see [Fagan76]) is a detailed technical review by peers of the authors of the software. A management review carries with it a strong concern for development schedule and allocation of organizational resources. Finally, an audit emphasizes independent evaluation of software quality. The difference between formal and informal reviews is that formal reviews involve product evaluation that is meant for more than the immediate use of the software author.

The distinction of the IEEE standard between between formal and informal technical review processes is a fundamental first step. Ledgard ([Ledgard87], pp. 65-70) discusses informal "work-reading" reviews. This approach and the review of a piece of software by its author are both important "technical review processes." However, the following paragraphs discuss formal group review processes.

Formal review reports serve these major functions:

1. They are used by software authors to remove defects.
2. They are used by management to assess project status.
3. They provide a historical record of software development.

The emphasis in the use of software reviewing in this course will be on peer group technical reviews. The diagram below relates the characteristics of technical reviews to walk-through techniques, which may be more familiar to you.

The goals of a technical review include:

- Cost-effective product enhancement by timely detection of defects.
- Personal growth and communication among software development professionals.
- Fostering teamwork, professionalism, participatory decision-making, and high morale.
- Improvement in the ability of reviewers to prevent defects in their own work. (Defect prevention is a major goal of any quality assurance technique.)
- Enhancing the effectiveness of testing by detecting errors prior to testing.

Varied complementary viewpoints should be a major goal in selecting the members of a software review group. The persons in a review group must represent the interests of everyone who may ever have a need to use the particular piece of software which is being reviewed. For example, a system specification must address all of the concerns of people who understand the system requirements, it must serve as a basis for subsequent phases of system development, and it must be testable and modifiable. This means that the review group should contain some expertise in the application area, technical expertise relative to subsequent development needs, and possibly specialized skill in testing or technical writing. Since students are unlikely to have all of this specialized expertise, student reviewers must make a conscious effort to try to analyze software as they think these different experts would see things.

In practice, the formal goals of a software review are to assess the quality of a piece of software and raise issues which might enhance or assure its quality. Software reviewing exercises are also used to provide learning situations in which participants develop their understanding of software development by detailed analysis and

COMPARISON OF A FORMAL MEETING AND A LECTURE

Formal Meeting	Lecture
• Technical Inspection Review	• Structured Walk-through
• 3-7 participants	• 1 person to large audience
• leader control high	• presenter leads, using scenarios
• producer participation passive [Fagan76] or none [Freedman82]	• producer participation high
• high participant preparation	• low participant preparation
• low demand on presentation skill	• high demand on presentation skill
• product must stand on its own	• presenter explains product
• covers limited amount of material	• much material can be covered
• uncovers many issues	• superficial detection of flaws
• high demand for group interaction	• low demand for group interaction

discussion of examples of software development documents. The immediate goal of a software review in either context is to raise issues about the software which is being reviewed. An issue is generally something which is inadequate about the software relative to its agreed upon goals, but it may also be something so remarkably important that it requires special mention. Reviewers should resist the urge to suggest how to resolve issues—that is the job of the software author. Reviews should raise issues, not resolve them.

Questions of style and minor errors should not be discussed in review meetings. Style issues are generally not specific to the software which is being reviewed, and they can cause a meeting to become bogged down in discussions which should take place in a broader context. Minor errors should simply be noted in the reviewed document.

Concerns about "correct" language use and document form are important, and poorly written communication cannot be accepted. Student reviewers tend to be especially sensitive to any instances of poor writing. Writing style may be a significant overall issue with regard to a particular piece of software, but minor writing errors should not be allowed to occupy the energy of a formal group review. The use of automated spelling and style checkers by the author can be very helpful.

The attached handout, "Software Specifications: Categories for Remarks" (see p. 66) lists examples of helpful types of issues.

The actual use of software reviewing in the practice of software development is quite varied. IBM makes extensive use of "Software Inspections," and Bell Labs is noted for rigorous walk-throughs. In general, our graduates are likely to see lots of informal walk-throughs, frequently involving only one or two persons, or members of the project team, which is not quite what the idea of group review by peers from outside the project is all about. Technical reviews may consist of a formal acceptance procedure as part of a "configuration management" process, with an individual or a formal committee which is charged with assuring the quality of

pieces of software before they are accepted by technical services people. The following list contains a number of specific benefits which have been attributed to software reviewing.

- Reduction in errors in the first production run of a one-line maintenance change (55% --> 2%, [Freedman82]).
- Reduction in errors in first production runs of maintenance changes (85% --> 20%, [Freedman82]).
- Reduction in seriousness of errors (anecdotal, [Freedman82]).
- Reduction in production crashes [illegible] first six months of operation of a system (77%, [Freedman82]).
- Reduction of 83% in field trouble reports [Freedman82].
- Reduction of errors in COBOL code (from 5.5 errors/thousand lines to 1.0 errors/thousand lines, [Freedman82]).
- Increased productivity through early detection and removal of errors, and defect prevention ([Freedman82], and statistics in [Boehm81]).
- Finding and removal of 30% to 70% of logic and design errors in typical programs, as high as 80% of program errors (IBM, [Myers79]).
- Higher test scores from students who use peer group review techniques on each other's code in programming classes [Lemos78, Shelly82].

Notes on the Software Development Life Cycle

The development, installation, and modification of many software-supported systems involves significant human effort, and defects in software can have serious consequences. In all but the most trivial software (software researchers use the term "toy programs"), identifiable types of recurring activities occur. The most common of these software development activities are frequently named and studied together in the context of software life cycle models.

A list of software development tasks which are often included in software life cycle models is attached to these notes (p. 10). The degree to which this list captures what software developers actually do or should do depends on three major factors: the size of the project, the application area, and the development context. Some activities may be omitted or involve minor effort; others may be done in parallel or in different sequence than what is listed; and still others may have to be redone as subsequent system development effort provides additional input.

The descriptions of the items in the list reflect the fundamental importance of the concept of abstraction to successful software development. Abstraction is simply the suppression of unnecessary detail. Software systems consist of layer upon layer of abstraction, including machine code, source code, operating systems, job control statements, and events in the application domain. Abstraction allows a complex task to be worked with in simpler chunks: it is the essence of the meaning of the word structured in the term structured programming.

In the context of Computer Science 319, *Software Engineering Principles*, the items in this list of software development activities provide a framework of things which must be intensively studied. Concepts of software quality assurance will be integrated throughout the course, with the exception of program testing, which is viewed as an implementation concern. The process of modifying existing software (i.e., software maintenance) is viewed as a special case of software development, since all of the listed software development activities can be considered separately for software maintenance tasks. The essential maintenance concept is that it is of overwhelming importance, and all software development activities must be designed to facilitate later modifications to software development products.

The concept of software prototyping will be treated separately because it incorporates much of the requirements-specification-design-implementation sequence of activities into a single activity. One use of prototyping is to provide an executable model which illustrates what might be done to serve user needs, and to a certain extent how these system behaviors might be implemented. The goal of a prototype is to illustrate a subset of system behaviors and functions at the user interface level, while avoiding implementation details as much as is practically possible. Prototyping is especially helpful in demonstrating system behaviors which are new to the user. A flexible prototype also allows system engineers and users to develop their understanding of what is needed and why. Prototypes are weakest at demonstrating performance characteristics of proposed systems (e.g., response time and interaction with existing systems).

The principal difficulty with software development and quality assurance is the dramatic increase of the difficulty of software development projects as they become more complex. In other words, when a task involves too many considerations, it requires an exorbitant effort and it involves a high risk of error. The hierarchical abstractions represented by the requirements-specification-design-implementation approach can be helpful because they structure the various types of relevant details into separate, distinguishable activities. This approach also facilitates two techniques for managing complexity: projection and partitioning. Projection in software development is the principle of simplifying a task by viewing it from different perspectives. For example, a library system can be viewed from the perspective of patrons, circulation, acquisitions, and cataloging. Partitioning is the principle of divide and conquer. For example, software which performs data entry and validation functions can often be developed separately from software which performs specific data processing functions.

If a software development activity appears to overlap two different life cycle phases, then the principle of abstraction is being violated by including unnecessary detail. This increases the complexity of that particular activity, and any future efforts to reconsider it, especially during system maintenance. Whenever system complexity is a dominant concern, which it often is in practice, software life cycle models provide essential guidelines for an organized approach to software development.

Life cycle models also help to keep priorities in order. Consider the popular remark that "When you're up to

your bleep in alligators, it's hard to remember that your original objective was to drain the swamp." In terms of a system life cycle model, draining the swamp is a system requirement, the system specification includes ditches specific drainage rates, the design states where the ditches will go, and the alligators are an implementation problem.

In the "real world" practice of software development, a life cycle model provides a useful way of planning, organizing, controlling, directing, and evaluating software development activities. In short, it supports the management of software development projects. Written records document the results of software development activities. The adoption of a particular life cycle model implicitly defines appropriate types of software documentation, together with an appropriate level of abstraction for each, and the audiences that this document must communicate with. For example, a software system specification must be understood by the authors of the software requirements, so that they can verify the product's completeness and consistency with their intent. The specification also must be understood by user clientele who are part of a particular approval process; it must state the external behaviors of the intended system; and it must provide adequate direction to the system designers who will work from it.

The amount and form of documentation is often subject to local standards extrinsic to a specific software development project, but the guiding principle for documentation should be that it must be *useful*. The difficulty in applying this guideline is that the authors may never realize who will use the document and for what reasons. This is why extrinsic standards and apparently arbitrary guidelines are needed. A helpful general guideline should be to document each life cycle activity as it happens and have that documentation reviewed before that particular life cycle activity is considered to be complete. The reviewing procedure needs to be formalized, complete with sign-offs and management reports, because of its importance. It also needs to be organized and carefully managed because of the difficulty of checking the correctness of something which cannot be "tried out" and fiddled with.

Because the process of software development is dynamic and context-dependent, additional experience with large projects is required in order to appreciate the complexity which is not apparent in the any software life cycle model. Computer science internships and Computer Science 320, *Software Engineering Practice*, provide some of this experience. The following list should be helpful. Please feel free to discuss this handout with your instructors.

Software Development Life Cycle Activities

1. Requirements Definition
   - Goals - Understanding of what is needed and why it is needed. A statement of a requirement is likely to have the form "The system must <statement of user need>." Words like *must*, *should* and *desirable* indicate prioritized requirements. General system goals and ideal system behavior may be included with statements of verifiable system requirements.
   - Input - Data collected about the problem domain, particularly from any existing systems, and the analysis of that data by "system analysts."
   - Output - A system requirements document, which states the system goals and specific needs to the best ability of system analysts and users (or their representatives). The rationale for specific requirements and a model of pertinent features of the application context should be included with the requirements document when they are appropriate for the needs of a particular project.
   - Control - Approval of users or their representatives, formal reviews of the requirements and derived software elements, especially a software system specification document and system validation testing.
2. System Requirements Specification
   - Goals - a statement of what a system that will meet the stated requirements will look like to the user (i.e., system behaviors at user interfaces). A system specification is stated as a verifiable system behavior that the "target system" will have. (Note : the writing of detailed "design specifications" which describe the operating constraints and behaviors of individual system modules is part of "system design.")
   - Input - System requirements, knowledge of the capabilities of the specified computing environment for the target system, and expertise in building computer-based systems.
   - Output - A document which states all relevant system behaviors and constraints, and verification procedures.
   - Control - Approval procedures and system testing.
3. System Design (High-level design)
   - Goals - A statement of how the specifications will be met.
   - Input - System specifications and system building expertise.
   - Output - System architecture and module specifications that are below the level of user interfaces.
   - Control - Approval procedures, module testing, system testing.
4. Implementation (Low-level design)
   - Goals - Program design, coding, testing, and debugging, together with system construction and certification.
   - Input - System architecture, functional specifications for programs, system data specifications, module interface specifications, system and program constraints.
   - Output - A finished product.
   - Control - Acceptance procedures and criteria.

5. Installation

- Goals - Training of users and operators and initiation of system operation.
- Input - System and user documentation, executable code, and an application context.
- Output - A smoothly functioning system which meets user requirements.
- Control - Acceptance testing, and post-installation evaluation.

Sample Software Defects

John A. Cross
Indiana University of Pennsylvania

Description: Examples of defects that might be encountered in a software technical review and how a review team might document them.

Objectives: To be helpful, sample defects should enable students to accomplish the following objectives:

- Learn what characteristics experts with experience and foresight consider to be defects.
- Identify similar defects in software artifacts.
- Document the defects which they detect in technical reviews.
- Make an effort to avoid injecting similar defects into software which they write.

Prerequisites: Students must understand relevant technical concerns for the type of software in which each sample might be found, the application domain of the sample, and the specific goals of the sample software.

Procedure: The samples may be used as lecture aids or as illustrative examples in textual materials given to students.

Evaluation: Students may be graded on their ability to detect similar defects and produce similar documentation. A software technical reviewing project may be used to evaluate student accomplishment of the above objectives in a specific context.

Commentary: One of the difficulties associated with detecting defects in technical reviews is the need to consider the details of a piece of software that is only a part of an integrated set of software artifacts. This software in turn constitutes the implementation detail of a software development project. It is difficult to provide short examples which are still meaningful after being extracted from a document that is suitable for technical review. The following examples are published in rough draft form because examples are essential to learning the underlying concepts, not because these particular examples are exceptionally good. Moreover, technical reviewers cannot limit their concern to types of defects which they already know. Rather, technical reviewers must strive to detect defects which extend their knowledge of what constitutes a defect. (Note: [Winograd86] discusses how language can limit our perceptions. An analogy can be drawn with the blindness that can result from overreliance on samples of defects or a reviewer's checklist.)

Sample Defects

ORIGINAL TEXT FOLLOWS.

Computer-Assisted IUP Degree Audit: System Requirements by Joe Smith and Jane Doe, 8/25/86

A. OVERVIEW

This paper outlines the development of a computer-based component of a system to provide computer support for IUP student advising. The computer-based component will be integrated within a single student records system serving multiple needs throughout the academic affairs division. At the same time, this component is not an end in itself; it must be integrated into the larger package offered in support of excellence in all dimensions of the IUP quest for quality. . . .

THE ORIGINAL DOCUMENT CONTAINED EIGHT MORE PAGES OF PROSE.

ORIGINAL TEXT FOLLOWS WITH REVIEWER COMMENTS IN ITALICS.

Computer-Assisted IUP Degree Audit: System Requirements by Joe Smith and Jane Doe, 8/25/86

A. OVERVIEW

This paper outlines the development of a computer-based component of a system to provide computer support for IUP student advising.

*[5-02]** *This is declared to be a requirements document. An "outline for the development of a system" should not be written until the requirements for the system are clearly documented and agreed upon.*

The computer-based component will be integrated within a single student records system serving multiple needs throughout the academic affairs division.

[1-04] *Cite a reference which describes this student records system in detail. A system which is integrated with the student records system cannot be specified until the student records system is clearly specified.*

At the same time, this component is not an end in itself; it must be integrated into the larger package offered in support of excellence in all dimensions of the IUP quest for quality.

[1-04] *This system is proposed as a subsystem of student records, which is declared here to be a subsystem of some unspecified larger system. This system cannot be required to be integrated into a system which does not even have a name, let alone specific form.*

ORIGINAL TEXT WITH COMMENTS CONTINUES.

*The numbers in these examples are taken from the attached list. See also the section, Categories for Defects in Software Technical Reviews.

Software Specifications: Categories for Remarks

ERROR CATEGORY	PROBLEM DESCRIPTION
*1-00	Missing/Incomplete/Inadequate
1-01	Criteria for a system decision missing or inadequate
1-02	Interface characteristics missing
1-03	Accuracy or precision criteria missing
1-04	Description of context inadequate
1-05	Processing specification missing
1-06	Error recovery specification missing
1-07	Missing emphasis
1-08	Data or process validation criteria missing
1-09	Acceptance criteria missing
1-10	Data specification missing
2-00	Inconsistent/Incompatible
2-01	Two or more specifications disagree
2-02	Incompatible with existing standards
2-03	Incompatible with existing systems
3-00	Unclear
3-01	Terms or acronyms need to be defined
3-02	Ambiguous wording
3-03	Muddled writing
3-04	Specification doesn't make sense
4-00	Extra
4-01	Outside of the scope of this project
4-02	Unnecessary detail
4-03	Redundant or wordy
4-04	Overly restrictive (includes specifications which are stated at too low a level)
5-00	Incorrect form
5-01	Typographical or spelling error
5-02	Writing error
5-03	Word processing error
5-04	Violation of standards
6-00	Incorrect technical detail
6-01	Specified processing inaccurate or imprecise
6-02	Specified processing inefficient
6-03	Specification not testable
6-04	Specification not modifiable
6-05	Description of problem or context incorrect
6-06	Technical error
7-00	General remarks

* Derived from a list in Bell, T. E., and Thayer, T. A., "Software Specifications: Are They a Problem?" *Proc. IEEE/ACM Second Intl. Conf. on Software Engineering*, October 1976.

Questions for Knowledge Assessment

John A. Cross
Indiana University of Pennsylvania

Description: Pretest of student understanding of software development phases and skill in writing software development prose.

Objectives: To assess student knowledge and skill. The results can be used to adjust course content to meet the needs of the students.

Procedure: These questions should be used before students begin studying the material in CM-3.

Evaluation: Answers and comments are printed in italics.

Commentary: Students must understand the concept of software development phases before they can be competent technical reviewers of software. The questions provide practice for students and knowledge assessment for their instructor. Although the questions have not yet been used for homework exercises or examinations, such use appears to be viable. The author welcomes comments from users of these materials.

Software Life Cycle Knowledge Assessment Form and Scoring Guidelines

General directions: Complete each item to the best of your ability. Your instructor is trying to determine how much you know in order improve the teaching of this course.

Part I. Identification
Directions: Label each item as an example of one of the following:

- a system requirement (REQ)
- a system specification (SPEC)
- a design detail (DES)
- none of the above (NA)

Include any comments that might help the instructor develop an understanding of what you know.

1. For an electronic mail system: The electronic mail system will be able to distribute a single message to up to 32 destinations with a single command. *<SPEC. The phrase "will be able to" is characteristic of SPECs. This statement expresses a specific system capability, not a user need.>*

2. For your dream house: All the living areas of the house must have connections for telephones and cable TV. *<REQ. The phrase "must have" is characteristic of REQs. A specification would say what kind of wiring, where the modular connections will be, what specific kind of connections, etc.>*

3. For a text editor: The virgule (/) will be used as a standard string delimiter. *<DES. A specification might state that a specific symbol will be used as a string delimiter, while the choice of that character is generally a design decision.>*

4. For an online grade book: Students must be able to check their grades by interacting directly with a host computer system. *<REQ. This statement describes a capability that the system must have.>*

Part II. Reviewing Software Requirements
Directions: Classify each of the following statements of software requirements as follows:

- acceptable as is (OK)
- needs minor revision (OKR)
- needs a complete rewrite (NOK)

Briefly explain why you decide on any OKR or NOK response.

5. For an appointment scheduling system: The system must be able to automatically schedule a meeting which does not conflict with the time schedules of its participants. *<OKR. The reasons for the OKR are constraint, preference, and exception handling (which might be addressed elsewhere in a specific software requirements document). This statement must be followed by a statement about constraints and preferences and what will be done when no solutions are possible.>*

6. For a text editor: The system must be able to find and display all instances of a user-specified character string in a text file. The display must indicate where the string occurs within the text file. *<OK.>*

7. For an online university student database: The system will store student identification numbers as packed-decimal integers. *<NOK. This is a design detail; it is entirely out of place in a software requirements document.>*

Part III. Reviewing a Software Specification
Directions: Classify the usefulness of each of the following software specification statements using the same categories with which you responded to the items in Part II. These statements are not necessarily connected with questions 5-7.

8. For an appointment scheduling system: The system will allow an authorized user to list all possible times for a meeting and then select one. *<NOK. A slight case of ambiguity: does the system pick one, or does the system*

user pick one?>

9. For a text editor: The system will have "global edit" capability; that is, a user may replace or remove all occurrences of a given string within a single document with a single command. *<OK.>*

10. For an online gradebook: Each course may have up to 32 sections, and up to 128 students per section. *<OK.>*

Part IV. Open-Ended Responses

11. Directions: For any two of the following computer-supported systems, write a one-sentence statement of a software REQ and a second sentence which states a related software SPEC.

- An online library catalog system for books (that is, a system to replace the card catalog)
- An electronic personal planning and appointment calendar
- An online university course registration system
- A medical diagnostic system
- A word processor

Sample answers:
REQ: The online catalog system must provide all the functional capabilities of the current card catalog.
SPEC: A patron will be able to search for a book by title or author.

REQ: The electronic calendar must be able to determine all possible times to schedule a meeting that does not conflict with the personal time schedules of any of the meeting participants. SPEC: The electronic calendar system will manage the personal appointments of up to 512 users.

12. Directions: For any two of the following software system needs, write a software REQ, a related SPEC, and a DES response.

- An operating system JCL
- An electronic mail system
- An online university registration system

Sample answers:

For an operating system JCL:

REQ: The user interface to the operating system must be user-friendly. <NOK: Goals are OK in requirements statements, but BE SPECIFIC.>

SPEC: The name for a system command may be abbreviated by any of a set of letters which agrees with the initial letters in exactly one valid command.

DES: The high-level command language parser will consist of a command identifier with appropriate error-handling, together with links to corresponding command processors.

For an electronic mail system:

REQ: The system should allow mail to be addressed by name.

SPEC: The system will be available 24 hours a day, 7 days a week. <OKR: This specification must be qualified—naively—by saying "whenever the host computer is up." A more detailed response might state a performance level, with alternatives to the mail system when the system is not available.>

DES: System users will communicate to the mail system by keying a one-digit command number which corresponds to a context-dependent menu of valid mail commands.

For an online university registration system:

REQ: The system must automatically satisfy all reasonable registration requests from students. <NOK: vague, not ble.>

SPEC: The system will reject a registration request for which a student does not meet all the prerequisites. This

restriction may be overridden by the course instructor keying in a special permission for the student.

DES: In order to guard against fraudulent registration, students may only register when they are logged on with an account that matches their current student id.

Guidelines for Teaching Concepts

John A. Cross
Indiana University of Pennsylvania

The following guidelines were developed at Indiana University of Pennsylvania during the teaching of the course, *Software Engineering Concepts*.

- **Practice:** Appropriate practice activity is essential to learning software technical reviewing procedures and the underlying theory.
- **Background Knowledge:** The successful implementation of any software reviewing activity requires that the reviewers have adequate knowledge of goals and guidelines for the software artifact they are reviewing. Reviewers need background knowledge of both the application domain and the particular software element under review; and these needs must be addressed by the instructor in a manner that is suitable to the goals of each particular software reviewing activity.
  - **Documents:** The need for background knowledge about the application domain requires that reviewers have access to the documents that are used to prepare the particular software artifact. In many situations, these documents are the principal means of communicating application-specific domain knowledge to reviewers. Thus, reviewers should have, for example, a specification and/or a design document for a program before they attempt to review the code. A viable supplemental or alternative procedure to providing documents to students is to have an application-domain expert available for consultation by the reviewers.
  - **Guidelines:** The reviewers also need guidelines for software documents. Fagan [Fagan86] calls these guidelines *exit criteria*. The instructor can communicate these guidelines through a combination of class presentations, printed guidelines for specific types of software artifacts, and checklists.
- **Goals & Procedures:** Students must understand goals and procedures for software reviewing before they can participate in a software reviewing activity. Classroom discussion should include: what reviewers should search for, how they should search, how they should report their findings, and how they should work as a group. The support materials in other sections of this document provide some examples of the things reviewers should report. Instructors may need to develop additional examples that are similar to the specific tasks they have planned for their students. Instructors should also thoroughly explain all procedures for recording remarks, interacting as a group, and reporting results. If computer-based procedures are used, instructors will need to verify that all students are literate in these procedures.

Part II: Implementing a Software Technical Review as a Learning Activity

Instructor's Checklist for Planning a Software Technical Review

John A. Cross
Indiana University of Pennsylvania

Description: Checklist to assist an instructor in preparing for a software technical reviewing activity by students.

Prerequisites: The user of this checklist should have a good understanding of the concepts documented in CM-3 and should have these support materials (SM-3) available for reference.

Procedures: The instructor should complete each item in the list before distributing materials to students to be reviewed, providing comments and document names were appropriate. The one exception may be item (8). This checklist has been found to be a useful planning guide. Software developers and educators believe that software technical reviewing procedures must be monitored constantly. This checklist should be viewed as an important part of that process.

Commentary: The author would appreciate hearing about any additions instructors have made to this list, as well as receiving comments on its use.

Instructor's Checklist for Planning a Software Technical Review

Directions: Each of the following concerns must be addressed by the organizer of a classroom software reviewing activity. The instructor may find it helpful to write specific statements or references to the documents that satisfy each item. References in square brackets refer to pages in these support materials that address each checklist item in a generic way.

1. Activity ID (Name of task or type of material to be reviewed, and date).

2. Goals [4, 19]

3. Inspectable materials [39]

4. Student knowledge of guidelines for type of software artifact to be reviewed [15]

5. Student knowledge of application domain

6. Student knowledge of how to detect defects [12, 64]

7. Individual reporting procedures [48]

8. Group interaction and reporting procedures [48]

9. Assignment of individuals to groups, and roles and responsibilities to group members [28, 48]

10. Forms [24, 30, 36, 61, 64]

11. Grading Guidelines [33]

12. Schedule

Biographical Data Collection Form

John A. Cross
Indiana University of Pennsylvania

Description: Form and instructions for the collection of "biographical" background data about students in a software engineering course.

Objectives: Possible purposes for gathering data about students include the following:

1. Getting to know students: their abilities, needs, and expectations.
2. Correlating student behaviors or performance with biographical data.
3. Providing a basis for dividing a group of students into subgroups or assigning roles in group projects. The sample form that follows was designed with this objective in mind.

Procedures for Instructor: Rationale and directions for each item in the form are presented below. Item (2) is the only item which is not generic, but the significance of each item must be considered in the context of the instructor's purpose: the selection of data items and categories for each item are context-dependent concerns. (Refer to "An Algorithm for Dividing Students Into Groups," p. 28 for a detailed procedure for dividing a class of students into groups for a software reviewing activity based on information obtained with this form.) It is likely that the data on this form will have greater value to an instructor if they are put into machine-readable form. Individual instructors may devise keying specifications for their form, or implement an online data entry program.

The following directions and rationale are keyed to the numbered lines in the sample form.

1. Some sort of identification is required, so that the instructor can communicate with students. It is recommended that students identify themselves by name on this form because it allows for subjective judgments about which students work together well in groups. The computer-id may be useful in communicating electronically with students, or allowing the names of students to be hidden during a pass at dividing the students into groups, thus preventing subjective judgments from influencing decisions at any particular point. Groups seem to work better if they include both men and women, so a "Sex" item is included on the form. With relatively small classes, "Student Name" is usually adequate to determine sex. The "System-ID" can be used in a number of helpful ways:
   - To cross-reference the background data to additional data.
   - To provide a unique ID when names are inadequate.
   - To provide an impersonal identifier for each individual's data when objectivity or confidentiality is desired. In some institutions, it may be necessary to generate such an identifier.

2. The specific responses to this question indicate the knowledge and points of view which an individual might bring to a software reviewing task.

3. The instructor can adjust the ranges and the way the question is asked to suit the nature of the students in the class.

4. Students may need to be told that an incorrect statement to this question may result in poor decisions about who is in which group. Note that the number of credits, specific courses, and grade-

point averages work together as predictors of the performance of students in a software reviewing activity.

5. The notes for 3 and 4 apply to this item and the next. This question addresses the need to determine the technical ability of the student to review a piece of software.
6. See the comments for item 5.
7. A mix of different ages is likely to provide a better group review than a homogeneous group. The age groups can be adjusted to suit the students.
8. Outside responsibilities can be a major factor in limiting the effectiveness of an individual student in a group project. Adjust this question, or ask an additional question about distractions, depending on the students.
9. The previous note also applies to this item.
10. Geographical distribution of students, commuting factors, and experience can all be important factors. Questions about these and other factors may be added to the form at the discretion of the instructor. Note that application domain knowledge is especially helpful in this context.

Procedures for Students: Students may complete the data collection form in about ten minutes of class time. If these data are to be used for grouping decisions, reflect that need in your planning. Students should talk with h other or bring questions to the instructor, so that any difficulties can be resolved. Note that this type of data also be gathered by an online system. However, allowing students to take a form home and submit it later invites hassles and incomplete data.

Evaluation: The performance level of individual students on software technical reviews can be predicted with a calculation such as the following. A nearly identical formula was 90% accurate at distinguishing high, average, and low performers for three iterations of the activity described in these support materials (with 65 students). However, the formula may be dependent on context, so use it cautiously (see p. 28). Note also that this formula predicts individual performance at detecting defects, not group performance.

Each blank in items 3 through 6 is assigned a value of 1 through 5.

prediction = 1 for each blank checked in item 2 (2 for interns)/2
+ item 3 * (item 4/5)
+ item 5 * (item 6/5)
- (item 8 + item 9)/10

For example, consider the following coded data for a 24-year-old female student with File Processing and Internship, 94 credits with 2.9 GPA, 24 CS credits with 3.4 GPA, working 12 hours per week, and taking 14 credits.

Mary Smith CKRLWP F 100100 4344212014

prediction = (1 [for file processing] + 2 [for internship])/2
+ 4*(3/5) (for overall course work)
+ 4*(4/5) (for computer science course work)
- (12 + 14)/10 (for distractions)
prediction = 4.1

Note: Sex and age codes are not used to predict individu performance, but they are used to measure how well group members complement one another.

If a statistical package such as SPSSX is available, the ographical data can be used for analyses such as the following:

1. Compare predicted with actual performance.
2. Examine alternative predictors of individual performance.
3. Analyze the data for correlations between background data and individual ability to detect specific types of defects in a software technical review.

Instructors should check prediction against actual performance and modify future procedures in light of this feedback.

Commentary: Every item in this background data collection form may need to be adjusted to suit the context in which it is used. If such data will be used to predict student performance, the user must keep in mind that the following factors are all fundamental to predicting the performance of an individual on a software technical review.

1. Technical ability relative to the particular type of software artifact under review.
2. Knowledge of the particular application domain.
3. Intensity with which the reviewer is involved in the technical reviewing activity.
4. Personality and leadership.

The data in the attached form can be supplemented with "psychological data" on cognitive style or personality type. A student of the author did this. The grouping decisions prompted by the psychological data supported the validity of the decisions based on the biographical data, but they did not provide any clear improvements.

SURVEY OF STUDENT BACKGROUND

1) Name _____ System-ID _____ Sex (M or F) ____

2) Please use a check mark to indicate the courses which you have taken (or transferred), and use an asterisk (*) for those which you are taking this semester.

____ File Processing ____ Database ____ Operating Systems

____ Internship ____ System Analysis ____ System Design

3) Check the appropriate category for approximate credits beyond high school.

____ 0-30 ____ 31-60 ____ 61-90 ____ 91-120 ____ > 120 credits

4) Check the appropriate category for approximate overall grade-point average.

____ < 2.0 ____ 2.0-2.49 ____ 2.50-2.99 ____ 3.00-3.49 ____ > 3.49

5) Check the appropriate category for total computing credits completed.

____ 0-10 ____ 11-15 ____ 16-20 ____ 21-25 ____ > 25 credits

6) Check the appropriate category for approximate Computer Science grade-point average.

____ < 2.0 ____ 2.0-2.49 ____ 2.50-2.99 ____ 3.00-3.49 ____ > 3.49

7) Check the appropriate category for your age.

____ 0-23 years ____ 24-28 years ____ 29-33 years ____ > 33 years

8) Average number of hours you will be working (for pay) per week during the time in which the software technical reviewing activities will be conducted: _____.

9) Number of credits you are taking this semester: _____.

An Algorithm for Dividing Students into Groups

John A. Cross
Indiana University of Pennsylvania

Description: Algorithm to assign students to review groups.

Purpose: Obtain balanced groups that work effectively, without penalizing individuals by assigning them to "bad" groups.

Procedure: See algorithm.

Commentary: The task of dividing students into groups for class projects can be approached in three basic ways:

1. Use a random selection process.
2. Allow students to choose their own group members.
3. Assign students to groups on some other rational basis.

The above approaches can be mixed. For example, an instructor could pick the best students as leaders, then either randomly assign the rest of the class to groups or allow project leaders to select their group members from the remaining students. The advantage of this approach is basic to any algorithm an instructor might use—some systematic consideration should be given to ensuring that each group has sufficient personnel resources to complete the project. A strictly random procedure incorporates a significant risk of producing unfair groupings relative to each group's ability to be successful on the project. Poor results might also occur if students are allowed to group themselves. For example, the best students might try to band together or students might group themselves on the basis of personal relationships rather than complementary abilities. However, the instructor may not be able to do much better, since individual abilities and group interactions are so hard to predict.

The procedure described below is more systematic than random. It utilizes data supplied by students, rather than relying solely on the subjective opinions of the instructor. The algorithm is based on two criteria for forming groups:

1. Groupings should be fair: there should be an even balance of total human resources for each project group.
2. Groupings should be heterogeneous: there should be a mix of different abilities, and complementary skill levels, sex, and ages.

The problem of determining groups is thus one of determining abilities and complementary personalities, then combining people into balanced groups which promise to be productive. Productivity is a key concern in this context because productivity may vary for a given group depending on the task or the working environment. The algorithm given below has been used to group students for a software reviewing task; it may or may not work for other tasks. However, the approach represents a starting point for a systematic approach to this problem in many situations.

The following procedure, which has been automated using a computer program, is a straightforward, flexible, and objective approach to the problem of grouping. The procedure is based on a biographical data form (see p. 24) and a personality profile form which attempts to obtain a rough indication of the cognitive style of each individual. Note that the algorithm described below is but one solution; other algorithms—or fine tuning of this algorithm—are also possible.

The Algorithm

Basic parameters: *number-of-students*, *group-size*

Input data: Biographical data and personality profile

1. Compute the number of groups so that, if there cannot be equal numbers of individuals in each group, the numbers will be as equal as possible.

   number-of-teams = CEIL(*number-of-students* / *group-size*)

2. Determine predicted abilities for the given project. (See p. 24.)

3. Sort the available individual data records on the basis of predicted abilities. (Use the point score for composite predicted abilities.) Assign the individuals from the top and bottom of the list to the same team, deleting them from the list, and repeat this procedure for each team until each team has two persons. Examine the resulting assignment for any imbalance in the number of males on any team. If necessary, adjust the assignments to balance the number of males in each group. Do the same sort of adjustment to provide heterogeneous mixtures of age levels in each group.

4. Repeat step (3) with the remaining individuals, adding one member to each group with each use of the list of unassigned individuals. With each new set of assignments to groups, adjust the assignments to balance the number of males and the heterogeneity of age brackets of the people in each group.

5. Print a summary report of team demographics; adjust manually if appropriate. Note the reason for the adjustment for possible inclusion in the algorithm.

6. Identify the group assignments with names, and make any needed subjective adjustments.

7. Reprint the summary report of team demographics and repeat steps (6) and (7) if the groups are poorly balanced or poorly mixed.

8. Look for leadership potential in each group. If a group does not appear to have anyone with leadership potential, further adjustments may be called for.

9. Decide whether to appoint group leaders or to allow another form of leader selection within each group.

Agreement and Release Form

John A. Cross
Indiana University of Pennsylvania

Description: Agreement for students to give consent for participation in research.

Procedure: If you plan to use the data gathered from classroom assignments for publication or as part of some other research, the students should sign consent forms. The following form can be used to obtain student permission to collect empirical data.

The instructor should acquire appropriate administrative approvals of this form before using it. (This form has been approved by two universities.) Have students complete the form and sign a paper copy prior to the collection of empirical data.

Commentary: Consent forms are a standard part of studies involving human subjects. Their use grew out of an ethical need for constraints on experiments using human subjects. Many institutions (both in academia and in industry) and publications require them. In some states they are required by law. In general, consent forms inform the subjects of the following:

- What they can expect.
- What harm (if any) they may incur. (In this case, will their participation have an adverse effect on their grades or what they learn from the course? It should not.)
- That they may choose not to participate or may withdraw at any time.

Undergraduate students of the author responded with respect and interest to the formality of this procedure. Use of this form may have caused a healthy "Hawthorne Effect," in which students performed better because they perceived that their performance was being more closely observed than normal.

Research Agreement and Release Form

In this course, your instructor will be collecting empirical data about your performance on your assignments. The special procedures that this entails are described on the following page. Please read the description of those procedures. When you feel that you understand what is planned, sign the agreement and release statement.

AGREEMENT AND RELEASE STATEMENT. By my signature on this document, I signify that I have read and understood the statement of planned research. Furthermore, I agree to participate to the best of my ability. The data which are gathered from this experiment may be published, but only in a manner which does not individually identify me.

SIGNATURE _____ DATE _____

Mailing address for a summary report of the results (optional):

_____

_____

_____

Questions may be addressed to :
<Instructor name>
<Address>
<Telephone number>
<E-mail address>

Research Procedures

You will be divided into groups of three or four students. Grouping will be based on your background, as you declared it on the "Biographical Data Collection Form," which you completed at the beginning of this course. Volunteers may be asked to serve as "coordinators" and "recorders" for each group. Each group will first review a software requirements document and then a software specification of verifiable system behaviors.

The reviewing procedure will consist of individual preparation of constructive comments about the software element which is being reviewed, followed by a review group meeting in which the group prepares a written report on the quality of the software. Two different procedures for coordinating the individual preparation of the review group members will be used. You will be graded on your individual mastery of the concepts of software development that are specific to the software which you review and on the quality of your group review reports. If the different grading procedures result in significant differences in your grades, an adjustment will be made to assure fairness.

In the course of these activities, your consistent, diligent participation is essential for our observations to have any value beyond this course. Therefore, NO LATE ASSIGNMENTS WILL BE ACCEPTED. If you are inadequately prepared for a review meeting, you will be dropped from your group. If you miss a class during the period in which you are learning about the software reviewing tasks or performing those tasks, you will not only compromise your ability to score well in this course, but you will also cause problems for your group and your instructor. PERFECT ATTENDANCE AND SINCERE EFFORT ON INDEPENDENT ASSIGNMENTS WILL BE DEMANDED OF YOU.

Alternative to Signing the Research Agreement and Release

You may elect to not sign this form. If you do not sign this form, or if you fail to meet group reviewing responsibilities, you will be placed in a review group with other students who are not part of the experiment. However, you must still do the same tasks and be graded on the same performance criteria as the students who participate in the experiment.

Initials of student: _____

Grading Guidelines

John A. Cross
Indiana University of Pennsylvania

Topic: Grading student performance of software technical reviews.

Objectives: These grading guidelines may serve several functions:

1. Provide specific direction for evaluating student performance.
2. Provide cues to students concerning outcomes that will be measured and the relative importance of each.
3. Provide feedback to students on the effectiveness and efficiency of both their individual analysis and group interaction.

The specific behavioral objectives that the grading guidelines address include:

1. Working individually and as a group, students will produce helpful output from a detailed inspection of a piece of software.
2. Students will be able to estimate costs and benefits in terms of both individual and group effort for a software technical review.
3. Students will be able to detect and document defects in a particular type of software.
4. Students will introduce fewer defects into software they create.
5. Students will value and know how to obtain the broader understanding that can be achieved by obtaining the views of different people and viewing the software from different perspectives.
6. Students will show enthusiasm for the use of software technical reviews as an effective means to evaluate the technical quality of a piece of software and document specific defects.
7. Students will show constructive and efficient group behaviors.

Prerequisites: The users of these grading guidelines need to know what constitutes a defect in the software being reviewed and how defects are to be documented. The software may include statements of known defects (see "Sample Inspection Material," p. 39). Instructors can depend on prior knowledge of any defects that are given, of defects detected by individual technical review of the document, or those deliberately inserted into the software before giving it to the students. In any technical review, it is possible that previously undocumented defects will be reported. If the software does not come with a statement of known defects, students are likely to report legitimate defects the instructor failed to notice.

Procedure: The defects documented by each individual must be evaluated by the instructor relative to how well they provide input to group interaction. Thus, a copy of each individual's annotations must be submitted for grading. For each concern an individual identifies as a defect, the instructor must determine whether the individual correctly identified a defect, documented a spurious concern, or made an error in his or her understanding of technical details.

Each issue must also be weighted for the significance of the underlying concerns and for how well it is reported. One approach is to assign an integer score of from -2 to +3 points for each issue raised. These scores are then summed for each individual. Since expert behavior is considered to be the correct identification of roughly 30% of the actual defects, performance considerably less than perfect should be awarded the highest possible score. The author recommends that (roughly)

$$highest\text{-}possible\text{-}individual\text{-}score = 0.30 * weighted\text{-}sum\text{-}of\text{-}known\text{-}defects$$

To grade group performance, the instructor must compare group output against the combined individual data

prior to group interaction. The following formula can be used for grading the defects reported by the group.

- -1 to +2 points for combining similar issues raised by more than one individual in a group.
- -1 to +2 points for editing the way an issue is reported.
- -3 to +2 points for removing an issue from the group report. The point value for this group action should be the negative of the point value for the underlying defect.
- -1 to +3 points for (synergistically) including an issue in the group report that was not in the report of any individual in the group.

The instructor must judge the relative merits of a particular group point total in the context of a particular task. Note that the amount of measurable "synergism" in the group report is likely to be low, even when there appears to be much group interaction. A score of +20 to +25 points may be a good performance.

The Summary Report and/or Related Issues Report must also be graded. The instructor can simply assign a possible point total and grade the summary reporting on that basis.

A possible set of class scores might look like this:

Individual	Group No.	Individual		Group		Group Reports	Composite Score
		Raw	Norm.	Raw	Norm.		
1	1	+12	+20	+11	+16	+6	+42
2	1	+18	+23	+11	+16	+6	+45
3	1	+10	+19	+11	+16	+6	+41
4	1	+6	+10	+11	+16	+6	+32
Possible		+62	+25	-	+25	+10	+60

Evaluation: The success of the grading process might be evaluated in terms of either the functional or behavioral objectives stated above. Both sets of objectives are important, and the simplest methods of evaluation are the most appropriate:

- Are these guidelines practical? (They worked for the author of this document. The experience of other users is now essential.)
- Are the resulting grades accurate? (Thus far, they appear to correlate with student grades on other tasks and in other courses.)
- Do these grading procedures foster the desired student behavior? (Experience with these procedures is too limited to assert that they offer significantly better learning advantages. However, they deal directly with the details of student performance or nonperformance, which is the essence of what should produce good learning outcomes.)
- Are there better procedures? (Certainly variations are possible, especially with the suggested point values and normalizing procedure. However, the details of determining the value added by both individual and group effort must be considered in any valid procedure.)

Commentary: The ability to automatically number, identify with initials, and merge student annotations is very helpful in this procedure. Computer programs to do this processing are relatively easy to construct. Instructors should consider a variety of software tools and statistical analyses. However, the following type of chart should be considered an essential method of data presentation for learning exercises in software technical reviewing ([Myers78] contains similar charts).

Defect Number	Reviewer Number								Points	%
	1	2	3	4	5	6	7	8		
1 (2)	2		1			2	2		7	48
2 (2)		3			3				6	25
3 (1)	1		1						2	25
4 (2)			2			2			4	25
5 (1)		1		1			1	1	4	50
6 (1)	1			1				1	3	38
Points (10)	4	4	4	2	3	4	3	2	26	33
%	40	40	40	20	30	40	30	20	33	

Student Opinion Form

John A. Cross
Indiana University of Pennsylvania

Description: Form to solicit student opinion about software technical review experience in the classroom ("debriefing" form).

Purpose: Obtain both open-ended and coded comments from participants in a software technical review. The form can be used to obtain data from the perspective of participants in order to diagnose problems or substantiate facts about the software technical review that are not otherwise documented.

Procedure: The form should be used after students have performed a software technical review and have submitted the group report. Students should be aware that their answers will not affect their grade.

The instructor may want to write a simple report to provide feedback to the students on the gist of their responses on this form. Public reporting should never identify an individual student, in accordance with the contract made in the Agreement and Release Form.

Data gathered from this form must never affect student grades. Summary statistics should be computed for coded items, which are organized in a way that makes them amenable to analysis by a statistics package such as SPSSX. Note that interrelationships might provide interesting insight, particularly with regard to group interactions. Other data that can be linked to the data from this form by means of the "System-ID" that also appears on other forms in these support materials.

Commentary: The coded responses have provided the most useful data.

OPINION SURVEY

System-ID _____ Group Review Role(s) _____

DIRECTIONS: For each of the following items, give your best opinion based on what you know at this point. Respond as if, in future reviews, you were selecting and using the most productive group interaction technique.

"Productivity" refers to a favorable relationship between costs (including time) and benefits.

- An "Average" rating indicates approximately equal costs and benefits.
- "Below Average" indicates that the costs appear to be greater than the benefits.
- "Above Average" indicates that the benefits appear to be greater than the costs.

Survey Rating Scale:
A) Superior B) Above Average C) Average D) Below Average E) Poor X) No Rating

_____ (1) Rate software technical reviewing as a way for you to learn about software development.

_____ (2) Rate software technical reviewing as a way for you to develop a breadth of knowledge of application systems.

____ (3) Rate software technical reviewing as a productive way for an organization to provide software quality assurance.

OPEN-ENDED RESPONSES (use the end of this form if necessary) :

(4) Say something positive about any or all of the software technical reviewing techniques used in this class.

(5) Say something negative about any or all of the software technical reviewing techniques used in this class.

(6) Say something, positive or negative, about using some form of software technical reviewing in other computer science classes.

(7) Which of the following group activities do you feel would be most helpful to you as a student as a follow-up to your first two reviewing activities in this course? Pick one (even if you have a preference which isn't listed) and comment on it. If your preference doesn't appear on the list, include a comment about it below.

(A) Review a "low-level" software element, such as a module design, or test plan, or code.
(B) Develop a "high-level" software element, like what you have reviewed so far.
(C) Develop a "low-level" software element.
(D) Develop a piece of software from a general problem statement through tested code (that is, a complete team project).

(8) General Comments

Sample Inspection Material with Key Remarks: System Requirements Definition

John A. Cross
Indiana University of Pennsylvania

Description: Artifact for a software technical review—a sample requirements definition and suggestions for the results of the review.

Objectives: To provide a document that has known defects and is appropriate for inspection.

Prerequisites: Students need to have an overview of the application and the goals of the artifact prior to their inspection activities.

Procedure: The instructor should determine the appropriateness of this document for the particular classroom context. The instructor may want to correct certain defects and/or insert additional defects.

Students must be provided with a copy of the requirements definition, which they can review for defects. Printed copy facilitates the review process; electronic copy facilitates the reporting process and group interaction. The recommended procedure is to provide students with both forms of the document.

See the other sections of these support matials for additional procedures for administering a software technical review.

After all technical review output has been reported, the instructor should provide detailed feedback, including: (1) what defects were missed and (2) what remarks were spurious or wrong, and (3) how productive the group interaction was.

Evaluation: The suggested results of the review follow the requirements document. The instructor should compare the students' reviews to these suggested results.

Each paragraph in the requirements document is followed by a paragraph number in brackets. The suggested results are keyed to those paragraph numbers. Some of the suggested remarks are identified with an asterisk, denoting a very important defect, or with two asterisks, denoting a critically important defect.

Commentary: Practice materials are generally hard to find, so it is necessary to take precautions in order to reuse the material in this text for subsequent classes. Two procedures are recommended. First, explain to students that they should not aid future users of this document. Second, edit the document with each use so that it contains a different set of defects.

IUP Learning Center Tutorial Program Scheduling

Project-ID: TUT01
Document-Type: Requirements Definition
Document-Status: Software Inspection Copy
Author: John Doe
Date: August 1986

I. Introduction

The IUP Learning Center Tutorial Program, located in 209 Pratt Hall, provides tutoring and study skills counseling services to any IUP student. Currently, tutorial services are offered in 18 disciplines (see SUBJECT listings in attached pamphlet) through approximately 25 tutors. Both of these numbers are expected to increase in the future. *[1]*

Each tutor is assigned to one or more disciplines. That is, a tutor may tutor in more than one subject area. In such cases, one of the assigned disciplines is identified as the "primary" assignment (others as the "secondary" assignments) for that tutor. Study skills counseling is considered to be a distinct discipline to which a few tutors are assigned. All the tutoring/study skills counseling takes place in Pratt Hall. *[2]*

The "Scheduling Desk" carries out all scheduling and day-to-day administrative activities for the Tutorial Program. The main job of the Scheduling Desk is to match up the requests of students asking for tutoring (referred to as "tutees") with appropriate tutors. Each tutor works according to his/her schedule, which is set at the beginning of the semester. Usually, tutoring is done on an individual basis, but some tutors allow more than one tutee to be present. *[3]*

The IUP Learning Center would like to computerize the scheduling of tutorial appointments to increase the efficiency of the Scheduling Desk and provide better service. This document describes the current system employed at the Scheduling Desk, the problems with the current system, and the desired system to be developed in the future. *[4]*

II. Current System

The scheduling system currently in use is a manual one. A tutee calls or stops by the Scheduling Desk and requests a tutorial session. The tutee will be asked to provide the name of the discipline, the desired day and time of the tutorial session (appointments may be taken up to two weeks in advance), and the tutor's name, if a particular tutor is requested. The worker at the Scheduling Desk searches for an appointment in a "Scheduling Book," a large binder that contains the schedules of all the tutors. These schedules consist of sheets of paper, one for each tutor for each day he/she is scheduled to work (see attached sample). On the sheet, the tutor's work hours for that day are indicated, and space for recording appointments is provided. These schedules are grouped by the discipline to which they are assigned. In case the tutor is assigned to more than one discipline, his/her schedule is placed under the discipline identified as his/her primary assignment. If the tutee's request cannot be satisfied, the tutee is asked if he/she has an alternative request in terms of tutor, day, or time. When the tutee's request is matched up with available tutor's hours, the tutee's name, phone number, and name of the course (e.g., EC 121) are taken and written down on the scheduling sheet. This indicates that the tutor has an appointment. *[5]*

The appointment may be cancelled, but it must be done so at least 24 hours before the appointment time. Otherwise, the cancellation will be classified as a "no-show." No-show is also detected when a tutee fails to keep an appointment. Two no-shows will disqualify the tutee from receiving any more tutorial services for the rest of the semester. The Scheduling Desk is responsible for keeping track of the name of the tutees who are

no-shows, and the number of times each of them does so. *[6]*

the day of the appointment, the tutee comes to the Scheduling Desk, where the appointment is confirmed. After the confirmation, the tutee is asked to fill in the "sign-in" sheet (a sample is attached). The information recorded on the sign-in sheet consists of the tutee's social security number, his/her name, and the course to be tutored. This sheet is for record keeping and later use in the analysis of the effectiveness of the Tutorial Program by both the Learning Center administrators and University administrators. Then the tutor for that appointment is called in to meet the tutee, and the tutorial session begins. Walk-in requests for tutoring are accommodated if tutors are available at that time. However, no record of walk-ins appears on the Scheduling Book since no appointments were made for walk-ins. Therefore, the walk-ins are recorded only on the sign-in sheet. When the tutorial session is over, the tutee comes to the Scheduling Desk to "sign-out." This is done by the tutee by filling in the column in the sign-in sheet that contains the duration of the actual tutorial session, counted in minutes. *[7]*

A tutor may change his/her schedule. These changes are accepted unless they affect appointments that have already been made. *[8]*

III. Problems with Current System

1. Some information is taken repeatedly. The information that is collected at the time of making an appointment overlaps with the information taken at the sign-in time. This is not only inefficient, but it also causes congestion around the Scheduling Desk by tutees waiting in line to fill in the sign-in sheet. *[9]*

2. Procedures for making an appointment for tutors who are assigned to more than one discipline are not efficient when the appointment is not for the primary assignment. The worker at the Scheduling Desk must flip pages of the Scheduling Book to get to the section where the tutor's schedule is placed. This situation occurs surprisingly often, especially when all tutors who are assigned to the discipline are filled up to capacity, and tutors who have a secondary assignment to that discipline must be checked. *[10]*

3. Procedures for confirming the appointment when the tutee comes in are inefficient. This is especially due to the fact that all the appointments start on the hour or half hour, resulting in most of the tutees coming in at the same time. Class schedules also influence these peak periods of activity at the Scheduling Desk. *[11]*

4. Checking for no shows is neither efficient nor reliable. This is because the worker at the Scheduling Desk must cross check the times and names in the Scheduling Book against the times and names on the sign-up sheet in order to determine who failed to show up for an appointment. *[12]*

5. The record on the sign-in sheet is organized in order of arrival time, but an alphabetical ordering would be easier to refer to. Also, use of the sign-in sheet results in the loss of confidentiality concerning the information on the tutees, since all the tutees use the same sign-in sheet where they can easily see the information on the tutees who came in before they did. *[13]*

6. Currently appointment restrictions set by tutors are not observed by personnel at the Scheduling Desk. For example, a math tutor, for one reason or another, may not feel comfortable helping someone taking Calculus II, and thus instruct the Scheduling Desk not to take appointments for that course. Another example is that some tutors who do accept group appointments may require that all the tutees who would be present in one session to be from the sections taught by the same professor. These appointment restrictions are not observed well. *[14]*

7. There are no procedures for recording which worker at the Scheduling Desk took which appointment. This caused some inconvenience in the past when disputes arose about the time/day of an appointment, or the existence of the appointment itself. *[15]*

8. The Scheduling Book is not updated on a daily basis. The schedules that contain the appointments for the day should be removed at the end of the day and new schedules that contain appointments for two weeks from the current day should be inserted. This is done only once a week simply because the procedure takes some time, and the Scheduling Book cannot be kept away from the Scheduling Desk for that time everyday. Workers at the Scheduling Desk also dislike the "flip-flopping" of the pages of the bulky Scheduling Book. This problem may be reduced somewhat by devising a new more efficient manual system instead of computerizing the Scheduling Desk operations. *[16]*

IV. System Goals and Requirements

As mentioned in the last section, it may be possible to improve the efficiency and effectiveness of the Scheduling Desk by modifying the current manual system. However, the whole operation of the Learning Center, of which the Tutorial Program is a part, is expected to go through the transformation from a collection of manual systems to an integrated computerized system. It is desired that operation of the Tutorial Program be computerized, not only for its sake, but also as a part of a larger, integrated system. *[17]*

Though the Scheduling Desk is only a part of the Tutorial Program, it is where most of the data concerning the Tutorial Program are collected, and thus the computerization of the Scheduling Desk will be a major part of the computerization of the operations of the Learning Center. The operations of the Scheduling Desk will be the first major manual system to be computerized, and therefore, the system to be developed for the Scheduling Desk must be modifiable so that it can accommodate future changes in the requirements. Future changes to the system are expected to concern the addition of new functions to the system, and the form and content of the output file created by the system for use by other Learning Center operations. Current goals and requirements for the system to be developed, together with a rationale for their inclusion, are presented below. *[18]*

1. The system must be highly user-friendly. — The people using the system cannot be expected to have any computer experience and the turnover rate of the personnel will be relatively high. *[19]*

2. The system must be highly robust. — For the same reason as specified in requirement 1, the system should respond to user errors by detecting them, diagnosing the cause, and offering helpful suggestions about how to correct them. *[20]*

3. The response time of the system must be reasonably fast, less than five (5) seconds is desired. — Quick response is crucial since the demands placed on the Scheduling Desk tend to be concentrated on certain times, namely a few minutes before and after the hour or the half hour. *[21]*

4. The system must be interactive and flexible enough to accommodate all the possible requests from tutees. — The most often heard requests from tutees are for: an appointment anytime during certain hours, an appointment with any tutor available, and/or a list of all the time slots during which a particular tutor will be available. *[22]*

5. Appointments should be scheduled for either 1/4, 1/2, or one hour, as requested by the tutee. The system must be able to accommodate any combination of such appointments. *[23]*

6. The system should be developed in such a way that all the necessary information on tutees and the courses in which they wish to be tutored are taken when the appointments are taken. — This would ease the confirmation of the appointment at the sign-in time since no new information must be taken at that time. *[24]*

7. The system must provide for easy confirmation of an appointment at the sign-in time. No duplicate entry of the data is desired. However, corrections to the information on the tutees and/or the course to be tutored may be made at this time. — The Scheduling Desk worker who takes the appointment might make mistakes in recording data. Also, a number of tutees do not know the exact name of the course in which they wish to receive

tutoring, thus giving the Scheduling Desk workers incorrect information. *[25]*

8. The system must be able to detect no-shows automatically, and keep track of all the no-shows throughout the semester. — When a tutee accumulates two no-shows, the system must either be set to automatically reject requests from that tutee in the future or print out the name of the tutee so that the Scheduling Desk workers will be informed of the disqualification of the tutee. *[26]*

9. The system must be able to handle cancellations. —Regardless of whether the cancellation is made within 24 hours of the appointment or not, the cancellation should make the tutor available for new appointments for the time of the canceled appointment. If the cancellation is made within less than 24 hours of the appointment, it should be classified as a no-show. *[27]*

10. The system must provide for usable back-up in case of a system failure. — Regardless of the hardware used, it is essential to have ways to schedule and meet appointments in case the system fails. *[28]*

11. Special restrictions placed by individual tutors must be displayed on the screen when the appointments are taken for those tutors. — The Scheduling Desk worker must see the restrictions before he/she takes an appointment for that tutor. However, the restrictions are to be displayed for the Scheduling Desk workers' information only, and the system should accept appointments that are not in accordance with the restrictions placed by the tutor. In other words, the Scheduling Desk workers must be aware of tutor restrictions, but they must also be able to override them. *[29]*

12. The system must be able to accommodate group appointments. — The system must allow more than one tutee to be scheduled for one appointment. Note that the two (or more) tutees may contact the Scheduling Desk at different times. Thus, the system must allow for scheduling another tutee at the same time slot as an already taken appointment. *[30]*

13. The system must be able to handle tutors who have multiple assignments. — The system is expected to free the Scheduling Desk workers from the job of memorizing which tutors have multiple assignments and which disciplines are their primary disciplines. For example, if a tutor who is assigned to French and Mathematics has an hour available for tutoring on Tuesday, that hour should appear as vacant regardless of whether the appointment is sought in French or Mathematics. *[31]*

14. The system must be able to make an appointment up to two weeks in advance at any time. — The computerized Scheduling Book must maintain and use complete, current scheduling data. *[32]*

15. The system must be able to accommodate both temporary and permanent changes in the schedules of tutors. — If the notification of changes is received more than two weeks in advance, the changes must be accepted and schedules must be modified to reflect those changes. If changes within the next two weeks are requested, the system must check for appointments already made that would be affected by the changes. If there are any such appointments, the request for schedule changes will be denied. Otherwise, the request will be accepted. *[33]*

16. The system must be able to display all the appointments a tutor has for the day. — The tutors very often ask the Scheduling Desk to show them all their appointments for the day when they come in for work. *[34]*

17. The system must be able to "block" appointments for all the tutors for certain hours and/or days. — There are times when no tutorial services are offered, such as holidays, the evening before the break, or the time when all the tutors are required to attend a Learning Center meeting. *[35]*

18. The system must produce a summary of the activities of the day at the end of the day or at the beginning of the next working day. — A file that contains information on tutorial sessions that took place during the day,

sorted alphabetically on the tutees' last names, must be created. Also, the system must be able to provide a daily report of the number of tutorial sessions that take place, grouped by the disciplines. *[36]*

19. The system is expected to offer security for the hardware as well as the software, including the data files. — The record of tutorial sessions as well as the record of appointments are confidential and should not be accessible to anyone who does not work at the Scheduling Desk, except the Learning Center administrators. Also, certain functions of the system, especially the ones that summarize the activities of the day, may not be accessible to anyone except Learning Center administrators. *[37]*

20. The system must record the duration of the tutorial session (measured in minutes) after the session is over. — Currently, the information concerning the number of minutes of the actual tutorial session must be input to the system retrospectively and somehow associated with the record of that tutorial session. *[38]*

21. The system must request the initials (or some other forms of identification) of the Scheduling Desk workers at each step of the operation. — It is desired that the system will not let the user proceed unless the user identifies him/herself. This is desired in order to keep Scheduling Desk workers highly disciplined as well as to offer some measure of security. *[39]*

22. When the appointment is made in person (i.e., the tutee comes to the Scheduling Desk to make an appointment), the system should be able to print locally a memo that shows the day and the time of the appointment, and the names of the tutor and tutee. — This is desirable, but it is not necessary. *[40]*

V. System Constraints

The system may be implemented on either the locally available mainframe or the microcomputers (IBM PCs) owned by the Learning Center. If microcomputers are to be used, the system will communicate to other computerized Learning Center operations by transferring its files to the mainframe through the use of a modem and a phone line. *[41]*

Suggested Review Results

De Remarks

1. *[1]* ** How much increase, and in what time frame?
2. *[4]* This document must "define requirements for a new system," rather than "define a new system."
3. *[5]* What if there is no match?
4. *[6]* * What data are recorded? How are files maintained? What reports or lists are needed? What volume of data is involved? What use is made of these data?
5. *[9]* ** Specifically which data are redundant? How do inconsistencies affect the overall system?
6. *[14]* ** Where are these restrictions recorded? What are these restrictions? Is some improvement in handling these restrictions a desired feature of the new system? How can a restriction be specified for a particular instructor's students, when there is no specification for how that restriction is recorded in the system?
7. *[16]* * The current operational detail is not clear. This document gives impression that appointments are entered as they are received. Also, what is done with old schedules when they are removed?
8. *[16]* The goal of this document is to state requirements for the new system, not patches for the old.
9. *[17]* * Who desires the computerized system? (The use of passive voice is a sign that something is not stated.)
10. *[18]* ** The total context is not yet defined, so integration criteria cannot be stated. Without an overall plan into which the proposed system it, the TUTOR system is premature.
11. *[19]* The term "user-friendly" requires precise definition.
12. *[20]* The term "robust" also requires precise definition.
13. *[20]* Requirements 1 and 2 overlap significantly. Combine them?
14. *[21]* * Response to what?
15. *[22]* * Only those requests that are documented at this point can be expected to be dealt with in the new system. Any additional types of requests may have to be built into the system as an (expensive) CHANGE.
16. *[24]* * What is "all the necessary info"?
17. *[25]* Requirements 6 and 7 are redundant.
18. *[26]* * What specific system response is desired: beep and print a message, produce periodic reports, flag a student against future appointments, cancel all existing appointments, ... ?
19. *[30]* * How many tutees max? Are there any room restrictions? Are tutees allowed to have special restrictions on group tutoring, or are there special restrictions by the topic area or tutee needs?
20. *[31]* * How does a time "appear as vacant"? How many areas of expertise may tutor have? What are the current areas of expertise, and how are they coded? What growth in this area is anticipated? What are the total effects of primary and secondary assignments, and is there any anticipated growth in this area?

[2] The meaning of "complete" and "current" should be stated. Also, note that "any time" could be construed to mean any time of day or night.

22. *[35]* The term "block" should be precisely defined.

23. *[36]* * This paragraph is "design constraining." Note: Are the reported data to be limited to tutee name, discipline, and time?

24. *[37]* * Exactly what are the security concerns: theft, vandalism, prying, hacking, unauthorized personal use of equipment, ... ?

25. *[39]* * The requirement to force system users to initial things is design constraining, and it may not be a good solution to the need for security and discipline. Also, items 19 and 21 overlap.

26. *[41]* Alternatives to file transfers are possible. Is there a particular reason for insisting on file transfers?

27. *[41]* ** Constraints on budget, development time, and development staff are not stated.

28. *[41]* * What are the estimated data volumes, processing requirements, and growth predictions?

Summary and Evaluation Report

(1) This document fails to define the context in which SOLAR will be required to function. System development should not proceed until this issue is resolved.

(2) User expectations for this system are unrealistic and incompatible. This problem must be considered when a high-level design is developed for an actual SOLAR system.

Status of software: NOK.

Part III: Implementing a Software Technical Review of an In-process Software Artifact

Directions for Software Technical Reviews by Students

John A. Cross
Indiana University of Pennsylvania

Description: Instructor procedures for administration of software technical reviews; handouts for students.

Prerequisites: In order to be effective at a group process for detecting defects, reviewers must have the following types of knowledge and/or skill. Student learning or skill development in the following areas may be more of a goal than a prerequisite, but the instructor should consider all of the following concerns in his or her planning. The concerns are:

- Knowledge of software application domain
- Technical knowledge of the type of software and its role in a software development life cycle
- Active strategies for detecting defects
- Writing skill to document defects
- Oral communication skill to participate in group meetings
- Knowledge of procedures and policy for a technical review project
- Knowledge of different points of view of participants in software technical reviews.

Procedure: Procedures are described on the next page. This, along with the various student handouts following should be read. Instructors can then create their handouts as required.

Evaluation: The procedures used for technical software reviews must be evaluated by management—in this case, the instructor. The following data can be used:

- Detailed grading data for both individual and group performance
- Historical data from previous reviews (and reviews in other contexts)
- A Student Opinion Survey (p. 36)

Commentary: The strengths and weaknesses of alternative procedures are listed and discussed in Deutsch and Willis [Deutsch88] and the "Notes on Software Technical Reviewing" (p. 4). Note that the software to be reviewed is an important element in making a software technical review effective. The fundamental concerns can be summarized as follows:

- Inspectable software
- Compatible knowledge of reviewer
- Complementary knowledge of group
- Active strategies for detecting defects

Procedures for Technical Software Reviews

Deutsch and Willis [Deutsch88] list five overall factors in achieving effective quality enhancement through ormal software technical review. The first three of these factors apply directly to all student technical reviews. These factors are:

- An overview of the technical reviewing task
- Individual preparation for group interaction (allow 4-7 days)
- Group interaction (allow up to 3 days to document group comments)
- Rework
- Follow-up

he instructor must distribute handouts to explain and document basic procedures and guidelines. These handouts must specify procedures for students, expectations for individual reviewing output, and expectations for roup activities. Basic materials are provided in this section, but there are significant alternatives that must be nsidered relative to the learning goals and work context for a particular software technical reviewing activity. he materials included in this section require the instructor to select one handout for individual reviewing rections and an additional handout for group activities. These materials do not exhaust the viable options, but ey do provide adequate direction to implement a valuable learning activity in most university learning situaons.

e suggested procedure for reporting comments by individuals and groups is to have students key in their m s if the environment allows it. The implementation of machine-readable remarks facilitates the pre- ation procedure, the group meeting, and the analysis of everything that happens in the review. This proved capability enhances the potential learning benefits that can be gained from the activity by facilitating analysis of its outcomes, and it decreases the paperwork. However, a manual procedure for recording ividual remarks can be a viable, organized approach to software technical reviews; the absence of a suitable st computing context should not affect the decision of whether or not to implement a software technical viewing activity in a particular learning context.

lect a face to face method of group interaction when students are uncertain of how to conduct themselves in a view. The computer-mediated group interaction procedure is most appropriate when an analysis of the group namics of the reviewing activity are a key concern for student learning or instructor research. Computer-diated group interaction also allows a geographically dispersed group of students to work together. Indiual response to the merged remarks of a group of reviewers is a convenient procedure because it takes less e and it does not require students to cooperate with each other. This individualized procedure provides nificant learning benefits with minimal cost, but procedures that involve group interaction are preferable cause of the importance of group effort to professional software development.

e choices for a specific technique for group interaction must be carefully considered—no single technique isfies all the technical review needs of current best practice. The Fagan [Fagan76] "Inspection" technique is most powerful procedure for evoking group synergism. However, it can fail if students do not understand otes of client, author, consumer, and tester. The Freedman and Weinberg [Freedman82] technical review cedure is less demanding (and less powerful). Both techniques require leadership from the review leader and rough records of the group's findings.

commended group sizes are: 3 or 4 for computer-mediated group interaction, 4 or 5 for a review meeting

[Freedman82], and 4 to 6 for an inspection [Fagan76]. In every case, the entire project should take no less than 5 days and no more than 14 days.

All technical review activities in an academic setting include the following elements. (See also the "Instructor's Checklist," p. 22.)

- Selection and dissemination of procedures for students
- Overview of software
- Individual software technical review
- Group interaction
- Instructor and/or student evaluation of student performance
- Feedback to students
- Instructor evaluation of each project
- Instructor recording of data on outcome of the activity
- Instructor monitoring of historical data

Student Handouts

On the next several pages are handouts or sections of handouts suitable for distribution to students. The instructor should choose from these as needed, depending upon the type of reviews being conducted and the administrative procedures appropriate to the educational environment.

General Reviewing Guidelines

The attached list of categories summarizes a lot of accumulated experience, so you may find it helpful to use it as an initial set of clues about what to look for. Your understanding of what must be stated in software development documents and the intended audience for various documents must be your basic guideline for what might constitute helpful things to say in a review. Your eventual goal should be to detect things worthy of a comment, write your comment, and then consult the category list to determine a classification for your comment.

You will be tempted to comment on the style of prose or code in the software. Comment on style only when the underlying meaning is wrong, ambiguous, extraneous, or incomplete. If a style standard for your particular software environment is clearly violated, document the situation as a defect. Other concerns about the style of the software reviewed may simply be listed as separate "Related Issues" (p. 64), but they are not to be reported as defects in the software. With regard to reviewing prose errors in spelling or grammar, they may be notes, but doing so will not contribute to the performance score for individuals or groups in this software technical review.

Classification of reviewer comments facilitates the analysis of review output, and it also seems to produce more thoughtful reviews. The "General Remark" category can be very important, since any list might blind the reviewers to issues not anticipated by the writer of the list. The primary concern in noting points about the software, being reviewed should be whether or not your comment helps the software document achieve its goals as well as possible for as long as it is needed.

You are expected to work independently and discuss this project only at the formally arranged times. Please report all activities that may be relevant to your performance on the reviewing tasks, so that your instructor can make correct inferences about the source of your fantastic performance. A time reporting form is attached [not included in these support materials] for reporting all time that you spend on this project. Please fill it out faithfully every time you do something relevant to this project.

Individual Preparation for a Software Technical Review

(0) Participate *actively* in an overview meeting.

(1) Study your handouts and notes until you have a clear idea of the goals of the document that you are reviewing. Review the types of defects that have been detected in past experience with similar pieces of software.

(2) Read a printed copy of the document that you are to review. Get a general feel for what it does and how it is organized. You will probably work more efficiently if you wait until your second reading before you make any extensive notes about defects, since some of your concerns will be resolved when you have read the whole thing.

(3) Reread the document and mark the points that you feel are worthy of a reviewer comment. You may want to

mark these points with a circled number for ease of reference. Write your comments on a separate sheet of paper. Make an electronic copy of the software that you are reviewing. Edit your comments into the document under the line that best points to each specific concern. Each comment must include a category from the attached list of categories for remarks about requirements [p. 66]. (Note that you may elect to record your comments directly into electronic form, thus skipping the transcription process. Do whatever seems most comfortable to you, but produce a document that has the same form as the sample.)

(4) Reread the document and your remarks. Check the software that you are reviewing one more time for such general things as verifiability, specificity, feasibility, and completeness. Compare the overall structure to your class notes and the particular guidelines that you will be given for each type of document that you review. Revise your comments as needed.

(5) Submit your edited copy of the reviewed document according to the guidelines in the "Document Submission Procedure" at the end of these notes. At the time when individual review comments are due, any student who is not ready will be excluded from the group review. This will hurt both the individual and the group, so *do not miss deadlines!*

Group Reviewing Procedure

Synergism is a key concept: the group review technique should yield results that are noticeably better than the sum of individual reviewer comments. This means that the group review should combine, filter, extend, clarify, and summarize the individual comments. In addition, new insight may be triggered by the remarks of other group members.

Your specific procedure is given on a separate handout, in order to allow you instructor the flexibility to choose the most appropriate procedure in your context. The required group document is a final form of the review comments (cf. "Sample Software Defect" [p. 12]), and a summary evaluation report of the software that was reviewed (cf. "Inspection Summary and Evaluation Report Form" [p. 61]).

Grading Policy Grading Policy for Software Technical Reviewing

In practice, software reviews should never be used for performance evaluation, since this could harm their effectiveness as a technique for achieving software quality goals. Managers should try to monitor the effectiveness of reviews, but they should never use them for performance evaluation. However, in software engineering, students will be evaluated on their performance on software reviewing tasks, since learning is the goal of the course. A composite grade will be assigned for each task as follows:

Individual comments	15 points
Group output	15 points
(each member of the group will receive the same score)	
Time reporting	2 points
Total	32 points
(Coordinator/recorder	2 bonus points)

Note two important things about grades. First, submission of your documents must be on time, as determined by their deadlines and the following "Document Submission Procedure." If the average group output scores for a particular reviewing technique appear to be significantly different from the other groups because of a particular

reviewing technique, your instructor. Your instructor may adjust the lower group scores upward in order to compensate for any unfairness in grading that might otherwise occur.

Document Submission Procedure

Mail your annotations to your instructor (alternatively, "your group review leader"). A single copy of the software with all of your group's annotations merged into it will be made available to you at your group meeting (alternatively, "at some specified time prior to the group meeting").

References

Copies of all references are available under your instructor's name at the library reserve desk.

Face-to-Face Technical Review Meeting

At the time when individual comments are handed in, your group must choose a coordinator and two recorders. Responsibilities for each of these roles are detailed below. Under the leadership of this newly selected coordinator, the group then chooses a time for a two-hour group meeting. Schedule this meeting within four days, since your preparation for the meeting must be fresh in your minds. Allow time after the group meeting for your recorder to prepare your final document before the due date.

Everyone in the group must participate in the entire group meeting in order to receive group reviewing points. Your instructor, or an official representative, will observe the meeting and make an audio recording of what is said. The meeting should take no longer than two hours.

Coordinator Responsibilities: The coordinator's first responsibility is seeing that the arrangements for the meeting are all taken care of. This includes seeing that everyone gets there on time. In a professional setting, the coordinator would also schedule a room, prepare copies of necessary materials, and see that all the review participants are prepared for the meeting. In this case, your instructor will take care of the room and the review materials. Any group member who fails to prepare for the meeting will already have been eliminated at the individual review submission time.

The coordinator's second responsibility is to act as a meeting leader. At the group review meeting, the coordinator should obtain the group's agreement on what must be accomplished in the meeting, and then see that the meeting goals are achieved as well as possible. This means that the coordinator should keep the discussion on track and keep an eye on the time. The coordinator must ensure that the individual comments are understood, merged, edited, or deleted, in whatever manner the group feels might be helpful. The old comments should be extended, and new comments generated whenever possible. Your group's final reports on the software that is being reviewed must state overall strengths and weaknesses and emphasize major issues, so be especially alert for main themes throughout the two hour period. If agreement cannot be reached on any point, consider the most negative opinion to be the review outcome, but report that there was disagreement on that point. The coordinator is also responsible for deciding whether or not to take a break.

The coordinator should not have to use heavy-handed or autocratic tactics, since the reviewers are competent, well-prepared (almost) professionals. The tools of the coordinator include a variety of group interaction tactics. The document can be reviewed in some structured fashion (top-down is generally better than beginning to end), everyone can be asked to say something good and bad about the product, one reviewer can be asked to blast it and another to praise it, or walk-throughs or other pseudo-simulations can be used. In this situation, we recommend that you consider the document as a whole at the beginning and end of the meeting, and take turns leading the discussion of individual paragraphs while you are going through the document from beginning to end.

The coordinator's final responsibility is to finish the summary report and obtain signatures before the meeting ends. At the conclusion of the meeting, the summary report should be submitted to the meeting observer.

Recorder Responsibilities: The recorder may be too busy at first to participate a whole lot. The recorder job will switch at break in order to allow him or her to participate more fully, and to spread the burden of preparing the final annotated document. Try to record in such a way that your notes are open to everyone at the meeting, and usable for forming the final reports. The technique of marking, numbering, and writing detailed remarks on a separate sheet of paper works reasonably well, especially with a single-spaced document.

After the meeting, a final version of your group's annotations must be prepared as a group document. The recorders are responsible for submitting this final document in the same manner in which they submitted their

individual documents. We recommend that you do this right after the meeting, but you have until the next regular class meeting.

Face-to-Face Software Inspection Meeting In the style of [Fagan76]

At the time when individual comments are handed in, your group must choose a coordinator, a reader/recorder, and individuals who will assume the points of view of author, client, and consumer. Responsibilities for each of these roles are detailed below. Under the leadership of this newly selected coordinator, the group then chooses a time for a two-hour group meeting. Schedule this meeting within four days, since your preparation for the meeting must be fresh in your minds. Allow time after the group meeting for your recorder to prepare your final document before the due date.

Everyone in the group must participate in the entire group meeting in order to receive group reviewing points. Your instructor, or an official representative, will observe the meeting and make an audio recording of what is said. The meeting should take no longer than two hours.

Coordinator Responsibilities: The coordinator's first responsibility is seeing that the arrangements for the meeting are all taken care of. This includes seeing that everyone gets there on time. In a professional setting, the coordinator would also schedule a room, prepare copies of necessary materials, and see that all the review participants are prepared for the meeting. In this case, your instructor will take care of the room and the review materials. Any group member who fails to prepare for the meeting will already have been eliminated at the individual review submission time.

The coordinator's second responsibility is to act as a meeting leader. At the group review meeting, the coordinator should obtain the group's agreement on what must be accomplished in the meeting and who will assume what roles, and then see that the meeting goals are achieved as well as possible. This means that the coordinator should keep the proceedings on track and keep an eye on the time. The coordinator must ensure that the comments documented before the meeting are understood, merged, edited, or deleted, in whatever manner the group feels might be helpful. Whenever it is appropriate, the old comments should be extended, and new comments generated. Your group's final reports on the software that is being reviewed must state overall strengths and weaknesses and emphasize major issues, so be especially alert for main themes throughout the two-hour period. If agreement cannot be reached on any point, consider the most negative opinion to be the review outcome, but report that there was disagreement on that point. The coordinator is also responsible for deciding whether or not to take a break.

The coordinator should not have to use heavy-handed or autocratic tactics, since the reviewers are competent, well-prepared (almost) professionals. The agenda is driven by the reader. Consider the document as a whole at the beginning and end of the meeting. Since the author is typically not very active in this context, have the author and read switch roles at some point.

The coordinator should submit the group's summary report at the conclusion of the group meeting. The coordinator must also see that the recorder submits a final set a detailed remarks.

Reader/Recorder Responsibilities: The "reader" in an inspection is responsible for reading the software, or paraphrasing it, one block (or paragraph) at a time. The reader notes any individually prepared remarks about each block after it is read. The coordinator then leads the discussion of that block while the reader/recorder makes a written record of all the concerns of the group.

The reader/recorder may be too busy at first to participate a whole lot. This job may switch at break in order to allow each reader/recorder to participate more fully, and to spread the burden of preparing the final annotated

document. Try to record in such a way that your notes are open to everyone at the meeting, and usable for forming the final reports. The technique of marking, numbering, and writing detailed remarks on a separate sl of paper works reasonably well, especially with a single-spaced document.

After the meeting, a final version of your group's annotations must be prepared as a group document. The reader/recorder(s) are responsible for submitting this final document in the same manner in which they submitted their individual documents. We recommend that you do this right after the meeting, but you have until the next regular class meeting.

Author: The author is responsible for knowing the application domain and all technical detail that has to do with the software under review. Reviewers may ask the author for detailed explanations of the software or the application context. The author may also ask the reviewers to clarify an issue that they want to raise. With a small group where the author is only "playing a role," the author may double as reader, and possibly even recorder.

Client: The client is the person responsible for the software elements upon which the software under review is based, or generally a software user. The person who assumes this role should be concerned with the validity and completeness of the software that is being reviewed.

Consumer: The consumer is the person who will implement the next step in the software development life cycle. This person should be concerned with the completeness, clarity, and verifiability of the software under review. In student inspections, this person can also play the role of tester.

Individual Reaction to Combined Group Comments

This technique may be used when other forms of group interactions are difficult to arrange, or for a *controlled study*.

At the time when individual review comments are due, schedule a two-hour period when you will work on your *group review* under the supervision of your instructor. During this time, you will produce a review that reflects the combined insight of your group, and its effect on you. Your instructor will provide you with a master copy of the review document into which he has merged the remarks of your group members. You will have two hours to react to your group's combined comments and prepare a summary group evaluation report (a form is attached to these directions). You will not have access to a terminal during this group review time.

Your role during this type of a group software review is really that of a *review coordinator*, but you have the option to add to the final review output in any way that you feel might be helpful. Since you are producing the only final output from the effort of the entire group, do your best to see that the software review goals are achieved as well as possible. Keep your thoughts on track and keep an eye on the time. Make sure that you understand the individual comments, then merge, edit, or delete remarks in whatever manner appears to be helpful. The old comments should be extended, and new comments generated whenever possible. Your final reports on the software that you are reviewing must also state overall strengths and weaknesses, and emphasize major issues, so be especially alert for main themes throughout the two hour period. If the individual comments disagree in an important way that you do not feel you can resolve by yourself, base your summary report on the most negative remark, but report that there was disagreement on that point.

After your two hours are over (or when you are ready), submit your summary report and a copy of your handwritten comments to your instructor, who will make a photocopy and give you back your originals before you leave. You have until the next regular class meeting to make your annotated changes in the master document for your version of your group's remarks. Submit these final "group remarks" in the same way in which you submitted your individual remarks.

Computer-Mediated Group Meeting Using Honeywell CB Software[2]

At the class meeting when individual comments are due, choose a coordinator and two recorders. (Responsibilities for each of these roles are detailed below.) Under the leadership of your coordinator, arrange two times for a 90-minute CB meeting, one in an optimistic time frame for completing your meeting preparations, and one in a pessimistic time frame. The point is that you want to have the meeting as soon as you are ready, but getting ready may take longer than you plan. You need to get your scheduling done in a face-to-face mode, because arranging a meeting time through MAIL can be very difficult. The total CB time should not go over 90 minutes, plus whatever time you spend practicing.

You will be using electronic mail, CB, and additional software tools to help you interact through the CP6. The idea of computer-mediated group interaction is to allow you to work in a *geographically dispersed* mode. You should decide on non face-to-face work sites to get a feeling for this mode of group interaction. If there is a problem with this ask your instructor for help (through electronic mail?). If you ever do find yourselves talking face-to-face, or working where you can see each other, please make a note of what happened in your time report.

In preparation for the group CB session, each group member must obtain a printed copy of the group's combined remarks. This document will contain the merged and numbered comments of your entire group. Your instructor will prepare this document as soon as possible, and adjust its attributes so that only the members of your group can access it: updates may only be made after your meeting, and then only by your group.

Each individual should mark his or her printed copy of the group's remarks with an evaluation of the helpfulness of each remark, according to the following rating codes.

5	Excellent, absolutely essential remark
4	Good remark, the group should keep it
3	Marginally helpful remark
2	Consider deleting this remark
1	Delete this remark
*	Discuss this remark in the group meeting.
Cnnn	Combine with remark number nnn.

Then, run VOTE (executable code) and enter your response to each remark. Your coordinator will prepare a meeting agenda from the results of your group's VOTEs, together with any mail comments that you send him or her. This agenda will be distributed to you through mail.

If any group member is unacceptably slow in working on this task, the group coordinator has the responsibility to declare that individual to be off the team (resulting in a score of zero for that individual's group review part of the assignment), or work something out so that the group is not hurt by that individual. Any group member who does not participate in the CB session will receive a score of zero for the group review. If the coordinator is not doing his or her job well enough, call in your instructor as soon as possible.

Your instructor will produce a computer-readable copy of your CB session, which your recorders will need.

[2]The CB program simulates a citizen's band radio communication group. Multiple online users may monitor and broadcast messages to each other.

Make sure that your instructor knows when the meeting will happen, and that he is maintaining a machine-readable copy of your dialogue before you begin. Please submit copies of all your project mail (through electronic mail, of course) to complete your group project records.

CB Protocol: The CB software could be a lot more helpful, but it can be functional. A few ground rules for group interaction should help a lot. The coordinator is in charge. When you want to say something, send "@". The coordinator will give you control of the communication medium until you have your say. End your message with "#" (or a 10-4?). Practice this procedure before you begin your 90-minute meeting in earnest. The coordinator may modify this protocol, but let your instructor know what you do and why.

Coordinator Responsibilities: The coordinator's first responsibility is seeing that the arrangements for the meeting are all taken care of. This includes all scheduling arrangements, and deciding what to do if something does not go as planned.

The coordinator's second responsibility is to act as a leader for the entire task. This includes writing an agenda, seeing that everyone prints and reads a copy of the agenda, and controlling the CB session. The coordinator must keep an eye on the time and keep the discussion on track. You will probably run out of time, so break in at an appropriate time and ask for everyone's overall impressions before you run out of CB time. You must prepare a copy of the Summary Report from the CB session, so make sure that you have everyone's remarks in the CB dialogue. Your group's final report on the software that is being reviewed must state overall strengths and weaknesses and emphasize major issues, so be especially alert for main themes throughout the 90-minutes of CB interaction. If agreement cannot be reached on any point, consider the most negative opinion to be the review outcome, but report that there was disagreement on that point.

Recorder Responsibilities: After the meeting, a final version of the review annotations must be prepared as a group document. The recorders are responsible for submitting this final document. The deadline for this is the time of the next regular class meeting. Prepare this document by editing a copy of your group's remarks according to the results of VOTE and the machine-readable record of the CB session.

Ask for whatever clarification you need during the CB session(s). When you write the final group comments, if something is still unclear, or the group is not in agreement, say so, and record the most negative group opinion. The recorders should not change the substance of what is decided in the group review meeting.

After the group interaction, all comments are from the group, not individuals, so do a global edit that removes the initials of the original author of the comment.

Software Inspection Summary and Evaluation Report Form

John A. Cross
Indiana University of Pennsylvania

Description: Forms for reporting of outcomes of a software technical review.

Objectives:

- In actual practice: to provide project management with a means of control over the project development process and to provide the software developer with a list of specific issues raised in the technical review.
- In a situation where the primary goal is to learn about software technical reviews:
  - to ensure that students perform all the specified technical review procedures.
  - to implement summary and evaluation reporting, an essential part of the software technical review process.
  - to force students to move beyond specific details to evaluate and summarize on a more general level.

Prerequisites: Students must be familiar with the technical review process and the procedure for work within each group.

Procedure: Each group receives a copy of the Summary Report form, along with a compilation of the comments of each member of the group. The instructor must emphasize that the form contains an essential agenda. It is the review leader's responsibility to follow that agenda and submit the group's final report. The instructor should stress that each item on the agenda must be conscientiously completed. The review leader should complete the form, but may delegate the task of writing the Summary of Findings section to a group member.

Two items on the form are particularly difficult: a summary of findings and an evaluation decision. Both items require group consensus on broad issues. The instructor should emphasize that participants in a review should be concerned not only with individual issues relative to specific points in the text of the software, but also with an overall evaluation of the quality of the software. The evaluation must be supported by a cogently worded summary. Review leaders should be instructed to assume the responsibility for allocating adequate time to agenda items (3) through (6). A minimum of twenty minutes at the end of a two-hour meeting is recommended.

The review leader is responsible for focusing the group's effort on the software under review. If an important issue raised in the review is not specifically or solely a defect in the software, the instructor may permit the group to attach a Related Issues Report as a separate item. Examples of related issues include notes about the meeting facilities or group procedures for the review. Some experts (e.g., Collofello in CM-3) recommend that related issues not be reported because the possibility of reporting related issues may distract the group from its primary concern—the software itself. The review leader should not allow extended discussion of any issue; the point of a group review meeting is to raise issues, not resolve them. This is a particularly true of related issues!

Evaluation: Technical review groups need specific feedback on their performance, both in terms of what they did or did not report and how well they reported it. However, the instructor may find it difficult to quantify how well each group completed its summary report. Realistically, the instructor can only grade (1) the completeness of the form and (2) the quality of the summary of findings and the related issues list (if one exists). The summary report should constitute one-fifth to one-third of the group's grade. Additional points may be awarded to the students who sign as review leader or recorder. If the group feels that one of its members did especially well or poorly in the group meeting, this feeling can be noted as a "related issue" and used as a basis for awarding a higher or lower score to that individual.

The person who evaluates the quality of a software technical review, based on the data reported on the attached form, should be sensitive to how the group allots its time to each agenda item and what non-agenda items receive significant group attention. The groups are likely to spend most of their energy on item (2) of the agenda, examining the line-by-line concerns of the group. However, it is important to learning—and, in practice, it is important to the project—for the technical review group to spend a reasonable amount of its energy on a broad view of the itemized findings. For this reason, the instructor may want to ask the group to enter the time of completion of each agenda item, rather than simply checking each one off.

Commentary: Students tend to focus on the details of the group's annotations of the software that has been reviewed. The group must have strong leadership to ensure that each item on the agenda receives adequate attention. Students are likely to be surprised at the differences between the group's report and their individual input, and between the instructor's "grading key" and the group report. These differences offer significant opportunities for learning, so the feedback process is important. The best groups achieve synergistic output, in which the outcome of the group meeting includes concerns not reported in the individual preparation of any group member for the group meeting.

The following form is one of several widely used forms. [Fagan76] has published forms which have been used at IBM sites; these forms include counts of defects in three categories. The author of this document selected a form that emphasizes agenda and minimizes counting and categorization. Error counts were omitted because of the automated processes used.

Note: Similar forms appear in [Freedman82] and [Yourdon85].

Software Inspection Summary and Evaluation Report

Project _____ Date _____

Review leader _____ Start Time _____ Stop Time _____

Agenda (Enter the time when completed)

_____ 1. Agree to functional roles (review leader, recorder, and (optional) reader); method of considering different points of view (user, producer, client, tester, quality assurance); review goals; and procedures.

_____ 2. Review merged comments: filter, combine, refine, and generate new comments.

_____ 3. Summarize major issues which have been raised by the group. List these major issues in the Summary of Findings.

_____ 4. Decide on the overall state of the software and report that decision on this form. Verify that the evaluation decision is supported in the Summary of Findings.

_____ 5. Attach to this report all documented review outcomes, including the group's edited annotations for the original software. If there is a Related Issues List, note that fact under the Summary of Findings.

_____ 6. Sign this report.

Summary of Findings

Evaluation Decision

____ Accept as is (OK).

____ Accept after minor revisions (OKR).

____ Unacceptable; revise and plan another review (NOK).

Signatures _____
(Note any
special role) _____

_____

_____

Related Issues Report Form

John A. Cross
Indiana University of Pennsylvania

Description: Related Issues Report Form.

Purpose: To provide a means for reporting significant issues that require the involvement of others, that cannot be resolved solely by the producer of the software artifact.

Prerequisites: To use this form, students must know what is considered a related issue.

Procedure: A brief explanation is included with the form. Students should be aware that this form is optional.

Evaluation: If this form is used, the data reported should be considered in the evaluation of the group's summary report.

Commentary: Related issues should be reported, but they should not be discussed. The review meeting must concentrate on the software under review—the coordinator (moderator) must not allow the attention of the group to be diverted to related issues. Sample related issues are given on the form.

RELATED ISSUES REPORT FORM

Project _____

Coordinator _____

Date _____

GENERAL GUIDELINES

This form may be used to report any concerns which are raised in a software technical review, but are not the appropriate or sole responsibility of the software author. Related issues generally fall into one of the following general categories:

- There is a defect in an artifact upon which the software under review was based. For example, a review of code may turn up an error in the low-level design of that code.
- Standards are in some way inadequate.
- Facilities, procedures, or participants are worthy of some special formal report. For example, it should be reported if too many participants were scheduled for the review.

NUMBERED LIST OF RELATED ISSUES

Categories for Defects in Software Technical Reviews

John A. Cross
Indiana University of Pennsylvania

Topic: Selecting a category for each software defect which is reported by a software technical review.

Objectives: Requiring reviewers to categorize each defect may significantly increase the difficulty of reporting defects. The objectives of the process must be clearly defined and the data must be monitored to assure that the objectives are being met. The benefits of reporting categories must justify the costs. The reasons for requiring software reviewers to categorize each defect include the following:

- To provide data which can be analyzed to determine facts about the software technical review process.
- To structure the documentation of defects and the corresponding thought processes which underlie that activity.
- To increase the amount of detail in the documentation of each defect.

Prerequisite Knowledge: If the software technical review process includes categories for each issue raised, reviewers must have a simple, clear set of standard guidelines. Reviewers must understand that the data they provide has significant value to their organization even though categorizing the issues is difficult and inexact at best. Reviewers must also accept that "management" (in learning situations, the instructor) understands that the categories may not be an exhaustive list of every type of issue that is appropriate to raise in a review.

Procedure: When providing a set of categories, the instructor should consider providing examples of issues for each category—issues which may be categorized in different ways depending on the point of view of the reviewer—and examples of the uses to which these data might be put by the organization requiring adherence to a specific set of standardized categories.

Reviewers should record categories within the text describing the defect. For example, the IBM procedure of categorizing defects as *missing*, *extra*, or *wrong* [Fagan76] can be implemented in an online reporting context by prefixing the text of each defect by an *M*, *E*, or *W*. Similarly, the [Bell76] categories, which are listed here, might be encoded with the error category enclosed in square brackets.

The person acting as recorder for the group has the final responsibility of assigning categories. Whenever a group of reviewers are involved, it is likely that different reviewers will assign different categories to the same defect (these data are considered unreliable). In order to avoid requiring the reviewers to resolve this disagreement, the instructor may want to allow recorders to assign more than one category to a single defect. For example, the following defect might be reported by a group using the [Bell76] categories:

[1-08, 6-01] The computation of interest earnings is based upon average daily balance, but the precision of this average daily balance has not been specified.

The review group should not spend time choosing the best category; that is why the task is assigned to the recorder. Categories which repeatedly overlap may indicate that the list needs to be refined or explained better.

Evaluation: It is difficult to assign scores to this activity in a classroom setting. Summary statistics should be computed, with an automatic process if possible. When the [Bell76] categories were used by the author of this document, statistical analysis of the results showed that different groups of students focused on different groups of categories. Points may be subtracted or added to the group score for group oversight, error, or helpful actions relative to categories. Individual students should not be penalized unless their categories are incomplete or grossly incorrect.

Commentary: The categorization process appears to be helpful for students, but the data have low reliability.

The author's students found the [Bell76] categories complex and difficult to apply. The [Fagan76] categories are even more complex because three separate categorizations are required. The following subset of the [Fagan76] categories is attractive because it is simpler and may be more reliable:
Alternative categories:

M, E, W for Missing, Extra, or Wrong
MAJ for Major or MIN for Minor

Each defect must be categorized both as M, E, or W, and as MAJ or MIN.

Software Specifications: Categories for Remarks

ERROR CATEGORY	PROBLEM DESCRIPTION
*1-00	Missing/Incomplete/Inadequate
1-01	Criteria for a system decision missing or inadequate
1-02	Interface characteristics missing
1-03	Accuracy or precision criteria missing
1-04	Description of context inadequate
1-05	Processing specification missing
1-06	Error recovery specification missing
1-07	Missing emphasis
1-08	Data or process validation criteria missing
1-09	Acceptance criteria missing
1-10	Data specification missing
2-00	Inconsistent/Incompatible
2-01	Two or more specifications disagree
2-02	Incompatible with existing standards
2-03	Incompatible with existing systems
3-00	Unclear
3-01	Terms or acronyms need to be defined
3-02	Ambiguous wording
3-03	Muddled writing
3-04	Specification doesn't make sense
4-00	Extra
4-01	Outside of the scope of this project
4-02	Unnecessary detail
4-03	Redundant or wordy
4-04	Overly restrictive (includes specifications which are stated at too low a level)
5-00	Incorrect form
5-01	Typographical or spelling error
5-02	Writing error
5-03	Word processing error
5-04	Violation of standards
6-00	Incorrect technical detail
6-01	Specified processing inaccurate or imprecise
6-02	Specified processing inefficient
6-03	Specification not testable
6-04	Specification not modifiable
6-05	Description of problem or context incorrect
6-06	Technical error
7-00	General remarks

* Derived from a list in Bell, T. E., and Thayer, T. A., "Software Specifications: Are They a Problem?" *Proc. IEEE/ACM Second Intl. Conf. on Software Engineering*, October 1976.

Checklists for Reviewers

John A. Cross
Indiana University of Pennsylvania

Description: Checklists for use in software technical reviews. The following checklists were derived from [Freedman82], [Bell76], and [Fagan76].

Purpose: To be used by reviewers to ensure that a specific set of possible defects is considered.

Procedure: These checklists may be used by student reviewers.

Commentary: The appropriateness of any checklist should be carefully considered by the instructor. Major considerations include:

- Is the particular checklist suitable for the software under review, the reviewers' abilities, and the procedures being followed?
- What defects are not covered by the list? Teaching note: students should know that any *a priori* list of possible defects can have the counterproductive result of blinding reviewers to defects that do not fit any of the categories on the list.
- Do the students understand the listed items and how to inspect for them? Teaching note: examples of the defects and practice in defect detection are recommended. The sample examination materials (p. 15) provide an indication of the type of materials that can be used.
- Is it appropriate to have reviewers categorize the defects they note in their reviews? Teaching note: the IBM procedure of categorizing each defect as something which is missing, extra, or wrong is easy to explain to students, but the effect of any categorizing procedure should be carefully considered. Individual reviewers must spend time and thought when they decide on a category and discrepancies between individual reviewers must be resolved during group interaction. (If the IBM procedure is used, it may be helpful to add a category for style annotations, which results in an acronym of MEWS for the complete set of categories.)

The following checklists are included in response to popular demand, not because they are highly recommended. Contributions of helpful checklists are needed. A helpful checklist must have proven validity in a significant breadth of technical reviewing contexts.

General Software Reviewing Checklist

Lists of attributes of quality software have been published in software engineering literature (e.g., [Boehm76] and [Freedman82]). The following checklist is an organized collection of criteria for quality software. By itself, this list may be too general for a thorough software review, but it can provide a helpful foundation.

Directions: Mark each point as "Not Applicable" (NA), "Acceptable As Is" (OK), "Correct with Minor Revisions Needed" (OKR), or "Unacceptable" (NOK). Cite specific problems by referring to numbered remarks and/or a specific point in the document being reviewed.

____ (1) CLARITY: The software must be clear, unambiguous, and precise.

____ (2) CORRECTNESS: The software must satisfy its specifications, and the final product must satisfy its requirements.

____ (3) COMPLETENESS: All required functions are fully implemented. All software is necessary and sufficient.

____ (4) CONSISTENCY: Design and implementation techniques, as well as notations, are uniform and in agreement with standards.

____ (5) TESTABILITY: All system functions can be empirically verified without inordinate cost. The software is safe.

____ (6) LEARNABILITY: The effort required to learn how to use the software is not unreasonable. Serious misconceptions are unlikely.

____ (7) USABILITY: The system is reasonably easy for its expected users to use.

____ (8) ROBUSTNESS: Computer programs, and the systems which they support, must continue to function according to specifications, even when small errors occur during their use. The system must appropriately respond to and recover from catastrophic abnormalities in its operating context.

____ (9) ERROR-TRAPPING: Computer-based systems must detect errors and deal with them appropriately.

____ (10) MODIFIABILITY: It should be possible to change the behavior or operating context of the software component of computer-based systems within reasonable limits and without unacceptable cost.

____ (11) OTHER DEFECTS: This category includes any additional deficiencies in the ways in which the software meets its requirements.

Software Requirements Reviewing Checklist

A software requirements document must state *why* a system is needed. It should focus on defining the problem and the context in which a solution must function. Software requirements include everything which is *required* for a system to be *useful*. Optional system features and ideal system functionality may be included, to the extent that it might influence subsequent system-related decisions. The goal of a software requirements document is to make a complete statement of a problem, together with known constraints on a solution. When they are relevant to the requirements of a software system, the requirements document must discuss the points listed below ([Abbott86] and [Sommerville85]).

Directions: Mark each point as "Not Applicable" (NA), "Acceptable As Is" (OK), "Correct with Minor Revisions Needed" (OKR), or "Unacceptable" (NOK). Cite specific problems by referring to numbered remarks and/or a specific point in the document being reviewed.

____ (1) The general rationale for a system development project—why a system should be developed.

____ (2) The conceptual model on which the the requirements are based—how the system will be used, together with other systems and procedures that will interface with the planned system.

____ (3) Data items that the system must handle, e.g., entities, attributes, and relations.

____ (4) Volume estimates for the data that the system must handle.

____ (5) Integrity constraints that the system must enforce, e.g., access limitations and error-handling requirements.

____ (6) Reliability and availability constraints, e.g., consequences of system failure and times when the system will be used.

____ (7) Legal constraints, e.g., privacy requirements.

____ (8) The knowledge and skill of system users, e.g., how frequently the typical user will interact with the system.

____ (9) Data processing functions that the system should perform, together with information needs of the user, e.g., which categories of system entities must be summed for reports.

____ (10) Hardware and software constraints, e.g., constraints on the peripheral devices or programming language.

____ (11) Response time requirements and expected system loading.

____ (12) Modifications that might be required later, e.g., likely changes in the hardware or software environment.

____ (13) Details of standard form, e.g., title, author, date, index, glossary of terms, local standards.

Software System Specification Checklist

A system specification should focus on *what* a system does to meet the system requirements. Only external behaviors are of direct concern. A software system specification must describe these behaviors in such a way that a system can be designed, built, and verified according to them. Thus, there are really two verifications involved:

- Does the specified system meet the requirements?
- Does the actual system meet the specifications?

A third demand is often placed on system specifications: a user or client may want to use the system specifications to decide whether or not to build a system or to determine which of several competing specifications to implement. The user or client may also want to modify the specifications. These uses for system specifications require a system specification statement that provides a basis for system understanding, verification, and detailed design. The General Software Reviewing Checklist is especially pertinent to system specifications. Additional criteria are listed below.

Directions: Mark each point as "Not Applicable" (NA), "Acceptable As Is" (OK). "Correct with Minor Revisions Needed" (OKR), or "Unacceptable" (NOK). Cite specific problems by referring to numbered remarks and/or a specific point in the document being reviewed.

____ (1) Is the specification clear, precise, and unambiguous for **all** appropriate audiences?

____ (2) Does the specification state what the system does to meet **all** of the system requirements?

____ (3) Does the specification provide an adequate basis for design or direct implementation?

____ (4) Are user procedures specified?

____ (5) Are installation, administrative, and operations procedures specified?

____ (6) Are behaviors specified for **all** system interfaces?

____ (7) Is there a specification of the total system supported by the software?

____ (8) Are system object types and instances specified? ("Type" includes constraints on value.)

____ (9) Are system data storage and processing limits specified?

____ (10) Are the system hardware and software environments specified?

____ (11) Are response time and system loading constraints specified?

____ (12) Are exceptions and anomalous inputs defined?

____ (13) Are reliability, error-handling, and system backup procedures specified?

Bibliography

James A. Collofello
Arizona State University

\ckerman83

\ckerman, A. F., P. Fowler, and R. Ebenau. Software Inspections and the Industrial Production f Software." *Software Validation, Inspection-esting-Verification-Alternatives: Proceedings of ie Symposium on Software Validation*. Amsterdam: lorth-Holland, Sept. 1983, 13-40.

Abstract: Software inspections were first defined by M.E. Fagan in 1976. Since that time they have been used within IBM and other organizations. This paper provides a description of software inspections as they are being utilized within Bell Laboratories and the technology transfer program that is being used for their effective implementation. It also describes the placement of software inspections within the overall development process, and discusses their use in conjunction with other verification and validation techniques.

The spection process at Bell Laboratories is pre d. Sample reports are included along with estimates for reviewing lines of code.

oehm76

oehm, B."Software Engineering." *IEEE Trans. omputers C-25*, 12 (Dec. 1976), 1226-1241.

Provides data on increasing error costs the later errors are detected and repaired in the software life cycle.

uck83

uck, R., and J. Dobbins. "Application of Software spection Methodology in Design and Code." *ftware Validation, Inspection-Testing-rification-Alternatives: Proceedings of the Sym-sium on Software Validation*. Amsterdam: North-olland, Sept. 1983, 41-63.

Another IBM variation of the inspection process for design and code is detailed. Sample reports are included. A discussion of how to interpret data as the result of review processes is also included.

eutsch88

eutsch M. and R. Willis. *Software Quality Engineering, A Total Technical and Management pproach*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

Table of Contents
Part I
Quality Concepts
Part II
Engineering-In Quality
Part III
Using Verification and Validation to Review-Out Defects and Test-Out Errors
Part IV
Management Aspects of Software Quality

A helpful presentation of concepts of software quality engineering. The chapters on formal technical review processes present strengths and weaknesses of alternative techniques for technical reviews by groups.

"Sample Software Quality Requirements Specification" (20 pages) in an Appendix provides an extended definition of software quality.

Fagan76

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

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

Freedman82

Freedman, D., and G. Weinberg. *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products*. Boston: Little, Brown, 1982.

This text is written as a series of questions and answers. It describes the Fagan methodology and many aspects of review processes. It provides a discussion of how to review many typical documents. It is weak in its discussion of sociological factors, review reports, planning issues and assessment of reviews.

Hart82

Hart, J. "The Effectiveness of Design and Code Walkthroughs." *Proceedings of COMPSAC '82.*

IEEE Computer Society's Sixth International Computer Software and Applications Conference. Silver Spring, MD: IEEE Computer Society Press, Nov. 1982, 515-522.

Many benefits of performing design and code walkthroughs are cited. Variations of reviews, including "round robin reviews," and their relative effectiveness are also noted. Actual sociological problems encountered at Sperry are also briefly mentioned.

IEEE80

IEEE Standard for Software Quality Assurance Plans. IEEE Computer Society Press, Silver Spring, MD, 1980.

The IEEE standard for Software Quality Assurance puts review processes into perspective with the entire software quality assurance process. Specific reviews are mandated by this standard.

McConnell84

McConnell, P., and W. Strigel. "Results of Modern Software Engineering Principles Applied to Small and Large Projects." *AFIPS Conference Proceedings of the 1984 National Computer Conference.* Montvale, NJ: AFIPS Press, July 1984, 273-281.

Abstract: This paper discusses the software development environment tools, techniques, and methodology as applied in two mediums to large real-time software projects. Both quantitative and qualitative measures of success obtained in these projects are discussed. The quantitative measures are statistics representing the size of produced code, the manpower over the project life cycle, and other data relevant to software engineering management. The qualitative evaluation is more concerned with results obtained from walkthroughs and various aspects of the applied methodology. Results are compared with those reported in the literature. Recommendations and suggestions for further improvements are presented.

The impact of review processes and their cost to implement on two medium to large real-time software projects are documented. The utilization of review processes to track a project is also described.

McKissick84

McKissick, J., M. Somers, and W. Marsh. "Software Design Inspection for Preliminary Design." *Proceedings COMPSAC '84. The IEEE Computer Society's Eighth International Computer Software and Applications Conference.* Silver Spring, MD: IEEE Computer Society Press, Nov. 1984, 518-519.

Abstract: The continuing need for improved computer software demands improved software development techniques. A technique for the inspection of preliminary software designs is described. Experience and results from the application of this technique are presented.

An inspection process at General Electric Company for preliminary designs is outlined including the roles of the review participants. The benefits of this process, including improved education, are also cited.

MILS85

Military Standard for Technical Reviews and Audits for Systems, Equipments, and Computer Software. United States Department of Defense, 1985. MIL-STD-1521B.

This standard defines the required reviews for military contracts. The appendices contain details about exactly what is to be covered for each of the mandated reviews as well as the role of the contractor and the contracting agency.

Peele82

Peele, R. "Code Inspections at First Union Corporation." *Proceedings of COMPSAC '82. IEEE Computer Society's Sixth International Computer Software and Applications Conference.* Silver Spring, MD: IEEE Computer Society Press, Nov. 1982, 445-446.

Abstract: At First Computer, a code inspection is conducted after the coding of a program or module is complete as indicated by a clean compilation of the program and prior to unit testing of the program. The completed program specifications and a clean compilation ar the entry criteria for the inspection process. An inspection team at First Computer consists of four members; one moderator and three inspectors. The moderator is the key person in theprocess with the responsibility to ensure the best possible review of the program. The moderator approves the team members for the inspection and makes the necessary decisions related to scheduling and conducting the sessions. The moderator is the facilitator of the inspection meetings but is also an active participant charged with finding defects. The moderator must log all defects found during the sessions, ensure that all defects found are corrected by the author, and decide whether or not to reinspect the code.

This paper presents a variation of the Fagan inspection methodology defining the process in depth along with the roles of the review participants. The benefits of utilizing their process are also documented.

Quirk85
Quirk, W. J, ed. *Verification and Validation of Real-Time Software*. Berlin: Springer-Verlag, 1985.

This text concentrates on testing techniques for real-time software. The utilization of review processes is also described. The emphasis of these review processes is, however, not unique to real-time software and very little insight into reviewing real-time systems as opposed to other types of systems can be obtained from this text.

Remus79
Remus, H., and S. Zilles. "Prediction and Management of Program Quality." *IEEE Proceedings of the Fourth International Conference on Software Engineering*. Silver Spring, MD: IEEE Computer Society Press, Sept. 1979, 341-350.

Abstract: Techniques such as design reviews, code inspections, and system testing are commonly being used to remove defects from programs as early as possible in the development process. The objective of the authors is to demonstrate that predictors can be devised which tell us how well defects are being removed during the defect removal process.

The paper presents statistical techniques for estimating the number of errors remaining in a product based on data collected from reviews. Approaches for evaluating reviews and the relationship of various reviews to each other and to testing are also described.

Remus83
Remus, H. "Integrated Software Validation in the View of Inspections/Reviews." *Software Validation, Inspection-Testing-Verification-Alternatives: Proceedings of the Symposium on Software Validation*. Amsterdam: North-Holland, Sept. 1983, 57-63.

Abstract: The software development process is looked at as to the specific contribution of inspections/reviews to the discovery of wrong design directions or implementations. The benefits are evaluated under the aspects of quality/productivity improvement and/or cost savings.

The relationship of review processes to testing in an IBM environment are explored. A variation of the roles of the review participants is presented as well. The utilization of defect data to the discovery of wrong design directions and implementations is also described.

Walker79
Walker, M. "Auditing Software Development Projects: A Control Mechanism for the Digital Systems Development Methodology." *Proceedings, COMPCON Spring*. Silver Spring, MD: IEEE Computer Society Press, 1979, 310-314.

Software audits and their function in a development organization are defined. Auditing techniques are presented as well as experiences from the Computer Science Corporation.

Weinberg84
Weinberg, G., and D. 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.

An overview paper describing the distinction between walkthroughs and inspections. The difference between formal and informal reviews is also clarified. The paper also contains sample review reports and how these reports can be used.

Yourdon78
Yourdon, E. *Structured Walkthroughs*. New York: Yourdon, Inc., 1978.

A detailed discussion of the walkthrough process. The benefits of walkthroughs as well as the mechanics of the process are presented. Psychological issues for walkthroughs are also noted.

Appendix: Addresses of Contributors

James S. Collofello
Computer Science Department
Arizona State University
Tempe, AZ 85287
CSnet: collofel@asu
ARPAnet: collofel%asu@relay.cs.net

John A. Cross
Department of Computer Science
Indiana University of Pennsylvania
Indiana, PA 15705
BITnet: jacross@iup
ARPAnet: jacross%iup.bitnet@vma.cc.cmu.edu

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS: NONE
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT: APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): SEI-SM-1-1.0	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: SOFTWARE ENGINEERING INST. / 6b. OFFICE SYMBOL (If applicable): SEI	7a. NAME OF MONITORING ORGANIZATION: SEI JOINT PROGRAM OFFICE
6c. ADDRESS (City, State and ZIP Code): CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213	7b. ADDRESS (City, State and ZIP Code): ESD/AVS HANSCOM AIR FORCE BASE, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: SEI JOINT PROGRAM OFFICE / 8b. OFFICE SYMBOL (If applicable): ESD/AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F1962890C0003
8c. ADDRESS (City, State and ZIP Code): CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
63752F	N/A	N/A	N/A

11. TITLE (Include Security Classification): The Software Technical Review Process

12. PERSONAL AUTHOR(S): John A. Cross, Indiana University of Pennsylvania

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
FINAL	FROM _____ TO _____	April 1988	76

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GR	technical review walkthrough inspection

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

These materials support the SEI curriculum module SEI-CM-3 "The Software Technical Review Process."

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☒	UNCLASSIFIED, UNLIMITED DISTRIBUTION

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
JOHN S. HERMAN, Capt, USAF	412 268-7630	ESD/AVS (SEI JPO)

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 Software Engineering Curriculum Project is developing a wide range of materials to support software engineering education. A *curriculum module* (CM) 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 (SM) contains materials related to a module that may be helpful in teaching a course. An *educational materials* package (EM) contains other materials not necessarily related to a curriculum module. Other publications include software engineering curriculum recommendations and course designs.

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.

Comments on SEI educational materials and requests for additional information should be addressed to the Software Engineering Curriculum Project, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213. Electronic mail can be sent to education@sei.cmu.edu on the Internet.

Curriculum Modules (* Support Materials available)

CM-1 [superseded by CM-19]
CM-2 Introduction to Software Design
CM-3 The Software Technical Review Process*
CM-4 Software Configuration Management*
CM-5 Information Protection
CM-6 Software Safety
CM-7 Assurance of Software Quality
CM-8 Formal Specification of Software*
CM-9 Unit Testing and Analysis
CM-10 Models of Software Evolution: Life Cycle and Process
CM-11 Software Specifications: A Framework
CM-12 Software Metrics
CM-13 Introduction to Software Verification and Validation
CM-14 Intellectual Property Protection for Software
CM-15 Software Development and Licensing Contracts
CM-16 Software Development Using VDM
CM-17 User Interface Development*
CM-18 [superseded by CM-23]
CM-19 Software Requirements
CM-20 Formal Verification of Programs
CM-21 Software Project Management
CM-22 Software Design Methods for Real-Time Systems*
CM-23 Technical Writing for Software Engineers
CM-24 Concepts of Concurrent Programming
CM-25 Language and System Support for Concurrent Programming*
CM-26 Understanding Program Dependencies

Educational Materials

EM-1 Software Maintenance Exercises for a Software Engineering Project Course
EM-2 APSE Interactive Monitor: An Artifact for Software Engineering Education
EM-3 Reading Computer Programs: Instructor's Guide and Exercises