Technical Report
CMU/SEI-88-TR-7
ESD-TR-88-008
July 1988

The Project Management Experiment

Peter H. Feiler
Roger Smeaton
May 1988

Technical Report

CMU/SEI-88-TR-7
ESD-TR-88-008
July 1988

The Project Management Experiment

Peter H. Feiler

Evaluation of Environments Project

Roger Smeaton

Resident Affiliate
Naval Ocean Systems Center

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Table of Contents

List of Figures

The Project Management Experiment

Abstract. This report covers a project management (PM) experiment, one of six experiments that examine different functional areas of Ada programming environments. The PM experiment was designed as part of the Evaluation of Ada Environments Project. This report describes the environment-independent part of the experiment: the activities covering the functional area, the evaluation criteria, and an experiment scenario to be performed on different environments. The experiment as it stands has been validated through internal and external review and through application to several environments that support project management.

1. Introduction

This report describes the design of a project management (PM) experiment that provides full coverage of project management activities. This experiment augments the set of experiments defined in the methodology for evaluation of Ada environments developed at the SEI [2]. In this methodology an experiment consists of two parts: an environment-independent description of a development scenario to be performed and evaluative criteria and questions, and the application of the experiment to different environments and analysis of the results.

The experiment design concentrates on the environment-independent part of the PM experiment. The design process has two steps, each of which has three components:

1. Designing the environment-independent part of the experiment, which involves:
    a. identifying and classifying project management activities
    b. defining evaluation criteria and establishing evaluative questions
    c. designing a generic experiment that is independent of a particular environment
2. Validating the design of the PM experiment, which involves:
    a. conducting internal and external reviews of the experiment design
    b. instantiating the experiment on several environments
    c. evaluating two environments based on the methodology including the project management experiment

The experiment design has been completed and the experiment has been validated both through review and application to environments. In addition to the original three environments evaluated by the SEI methodology, whose results are documented in [4], we have evaluated two environments (ISTAR from Imperial Software Technology, Ltd., and the R1000 Ada environment from Rational) with the extended set of experiments. The results of these evaluations are being published in 1988 as separate SEI technical reports. Experiences of using the SEI methodology are discussed in [5]. The insights gained in terms of requirements on integated project support environments as a result of applying the PM experiment to several environments is documented in [1].

This report first presents a brief review of the evaluation methodology (see Section 2). Section 3 describes the differences between the design and presentation of the generic PM experiment described in this report and the experiments discussed in the original report [4]. Section 4 defines the scope of project management covered by this PM experiment. The subsequent three sections (Sections 5-7) present the three components of the environment-independent experiment description. This is followed by instructions in Section 8 for the application of the experiment to an environment, based on our experience in executing the experiment in. The appendices contain support material for the application of the experiment and the recording and analysis of the results. Also included is a sample evaluation illustrating the use of the support material.

This report also satisfies the first two components of the validation part of the experiment in that the report has been revised based on reviewer feedback and on feedback from instantiating the experiment by a person who is not one of the two designers of the experiment. A full report of the evaluation of two environments (the Rational Environment from Rational and ISTAR from Imperial Software Technology, Ltd.), including the results of applying the PM experiment, is due the first quarter of 1988.

2. A Review of the Evaluation Methodology

For a more detailed discussion of the evaluation methodology, see Chapter 2 of [4], [2], [3], or [6].

The methodology developed to evaluate programming support environments is based on the following guidelines:

- Focus on user activities.
- Be independent of any actual environment.
- Be experimentally based (objective and repeatable).
- Emphasize primary functionality.
- Be evolutionary.
- Be extensible.

The evaluation methodology consists of six discrete phases:

1. Identify and classify software development activities.
2. Establish evaluative criteria.
3. Develop a generic experiment.
4. Develop an environment-specific experiment.
5. Execute the environment-specific experiment.
6. Analyze the results.

The first three phases are independent of any particular environment and are performed once for each functional area of programming support environments (design and coding, testing and debugging, configuration management, system management, and technical project management). The last three phases of the methodology are specific to the environment being evaluated and are performed once for each environment evaluated.

Figure 2-1 illustrates the relationship between the evaluation methodology phases. In Phase 1, underlying activities common to software engineering (the "what" of software development, not the "how") are enumerated. The criteria established in Phase 2 fall into four broad categories: functionality, user interface, system interface, and performance. Some categories will inevitably overlap, for example, functionality criteria and user interface criteria. The dominant category, functionality, is revealed mainly by checklists and descriptions of how activities are carried out on a particular environment. Note that in this PM experiment, the activities in a checklist are not given priorities or weights. We feel that judging which functions are more important (i.e., which should be weighted more heavily) and which are less important can only be done by the ultimate user of the results, not by the authors or the experimenter.

The results of Phases 1 and 2 are used to develop the generic experiment. The experiment is generic in the sense that it does not use specific tools in specific environments, but refers to generic tasks that must be performed. It must be detailed enough to allow the experimenter to instantiate it on a particular environment, but general enough not to imply a specific set of tools.

Figure 2-1: Phases and Products in the Evaluation Methodology

The environment-dependent part of the methodology (Phases 4, 5, and 6) involves instantiating the generic experiment on a particular environment, executing the experiment, and analyzing the results.

3. Refinement of the Original EAE Methodology

This Section summarizes the differences between the experiment descriptions in this report and those in the original Evaluation of Ada Environment (EAE) report [4].

In the original report, the methodology was developed and experiments were defined in five areas: design and coding, testing and debugging, configuration management, system management, and technical project management.[1] The purpose of the task described in this milestone report is to develop a project management experiment for the Evaluation of Ada Environment's methodology that provides full coverage of project management activities. Since this task was done after the methodology was applied to environments, we have attempted to improve the design of the experiment and the presentation of the material.

3.1. Level of Detail

The first category of differences is in the level of detail and the degree of environment-independence regarding activities, questions, and experiment steps. In the original report, activities, questions, and experiment steps were presented at a level of detail that was quite close to the set of primitive operations provided by the environments, which were evaluated in the same report. This level of detail raised some concerns because carrying out the experiment and answering the questions created a large volume of information. In the project management (PM) experiment, the volume of information would be compounded because the PM experiment covers a wider range of facilities than does the original set of experiments. Furthermore, the activities and experiment steps in the original report were, to a certain extent, environment-dependent because the level of detail was so close to the operations provided by the initial set of environments that were evaluated. As a result, Phase 6 of the methodology—the analysis of the experiment results—was more difficult to complete. The person performing the analysis had to abstract information from the detailed "raw result" data, and draw appropriate conclusions at a higher level of abstraction about how well the operations supported the user's tasks. Providing good guidance to the experimenter was difficult.

We dealt with these concerns by identifying activities and designing an experiment at a higher level of abstraction that more closely represents the *tasks* a user of the environment is trying to accomplish rather than the *operations* that may be provided by particular environments. For example, instead of specifying the activity "change ownership of files and directories," our scenario specifies only that a member of the development team leaves the team; the operations that are actually carried out by the experimenter to accommodate this change will depend on the environment being evaluated. From the activities, we built scenarios of situations that could occur in a real project; the scenarios are less dependent on any environment and, at the same time, reduce the volume of information collected by reducing the amount of detail. As a result, the reader gets a clearer overall impression of the capability of an environment that supports the tasks of a developer or manager.

[1]A sixth experiment—a compiler benchmark suite known as the Ada Compiler Evaluation Capability—was contributed by the Institute for Defense Analysis.

Our approach puts more responsibility in the hands of the person carrying out the experiment on a particular environment. The experimenter has considerable freedom in mapping activities and experiment steps onto operations in the environment, and sometimes has several alternatives for accomplishing the task. For example, in a given environment, specialized support may be available for submitting time sheets, reporting progress, and assigning tasks; or, the same activities may be accomplished by electronic mail. This may introduce more subjectivity into the evaluation because the experimenter does not have as much guidance in instantiating the generic experiment. Keep in mind, however, that the intent of the experiment is to determine the range and appropriateness of the facilities available in the environment. We expect the experimenter to describe the functional and architectural model of the environment, as well as describe how activities and experiment steps are mapped onto environment operations.

Another point worth noting is the secondary importance of performance metrics in the evaluation of environments. Performance has two aspects: *efficiency*, which considers both time and space requirements, and *responsiveness*, the appropriateness of the environment's response time to the complexity of an activity or command. In the PM experiment especially, we believe that measuring efficiency (i.e., milliseconds elapsed or bytes consumed) does not reveal much about the usefulness of the facilities in an environment. On the other hand, responsiveness is relevant and useful information to collect. Thus, approximate cost in time and space should be measured, but the experiment does not require the high accuracy and repeated measurements necessary to produce valid data points for the cost of language constructs.

3.2. Format of Experiment Information

The second category of differences between the original report and this report is in the presentation of the experiment information. We deviate from the format of the original EAE report in Phases 3, 4, and 5.

In Phase 3, there are two differences: First, the PM experiment must capture the activities of many people working simultaneously. Therefore, we describe the generic PM experiment by first laying out the context and the scenario and then describing each role as a time-ordered set of experiment steps. Second, the result of step 2 of Phase 3—associating questions with experiment steps—is not presented in tabular form. Instead, we list relevant questions and activities on separate lines of text immediately after each experiment step description.

The changes to Phases 4 and 5 affect how the experimenter documents the results of instantiating and running the experiment on a particular environment. In Phase 4, the experimenter describes the functional and architectural model of the environment and its ability to capture the experiment scenario, and describes the sequence of operations that must be carried out to accomplish the activities in an experiment step. The experimenter should work out the sequence of operations necessary to produce the actions called for in the experiment, but *not* necessarily develop a command script or keystroke file. We believe that the cost of developing such a script or file for the complete experiment for the sake of repeatability and precise performance measurements of a few operations does not warrant the effort, especially for highly interactive environments. Instead, the command sequence—the set of operations for each step —is validated by testing it manually from the keyboard and screen. With respect to performance, particularly the measurement of time, the experimenter should consider both the elapsed time to execute an operation and the time required to enter the information into the environment. Measuring both components will give a more meaningful metric of the cost of using a tool, since the cost of entering information through an awkward forms system, for example, may outweigh the cost of actually executing the operation. Finally, the experimenter completes the functionality checklist (see Appendix A).

Phase 5 in the PM experiment is primarily gathering performance data and answering questions. To document the results of this phase, we suggest that the answers to the questions be presented in the same order in which they were defined in step 2 of Phase 2, i.e., organized according to evaluation criteria. This makes the information more readable and understandable to both the reader of the report and the person carrying out Phase 6.

Phase 6 is essentially the analysis of results. For those activities that are supported by the environment, the experimenter summarizes the differences and similarities in the functional model of the generic experiment and the target environment and answers the question: How well are various activities from the generic experiments supported by the particular environment? If the experiment has been instantiated on more than one environment, a cross-environment comparison is also a product of Phase 6.

The functionality checklist to be used in Phase 4 (included here as environment-independent material), cross-reference tables that help organize the results of Phase 5, a global view of the experiment's scenario, and a description of MacProject to illustrate the architecture and functionality of a project management tool, are included as appendices.

4. Scope of Project Management

The generic project management experiment exercises the facilities for project management and product management of an Integrated Project Support Environment (IPSE).

The purpose of this experiment is to provide a comprehensive evaluation of an environment's project management facilities: project planning, monitoring, and control; instantiation of plans in the development facilities; management of development in terms of coordination and communication; and management of software products.

In the context of this experiment, the term "project management" covers activities of project managers (i.e., project planning and control) as well as management support for activities of developers. Because the coverage of project management activities is comprehensive, we do not expect any environment on the market to have all of the facilities probed for by this experiment. It will probably be necessary, therefore, for the experimenter to carry out a subset of the experiment. Section 8 includes hints to the experimenter about how to deal with incomplete coverage of the experiment.

The PM experiment is larger than the other experiments (design and coding, testing and debugging, configuration management, and system management). We expect that it will require two to three times the amount of effort on the part of the experimenter. Initial experience suggests that four to six man-months are required to carry out the full PM experiment.

The PM experiment simulates a project for maintaining a released software system. The "project" starts with a set of error reports from customers who are using the current release of the system. The error reports are analyzed, and a new release correcting many of the reported problems is planned. Project plans are drawn up by the project manager, together with the project leaders. The plans are approved and the project is carried out by several teams, as well as a documentation group and a quality assurance group. During the life of the project, its progress is monitored. Various changes (e.g., in personnel) are necessary to accommodate the execution of the project. Finally, the new release is made available for distribution to the affected customers.

The challenging part of the PM experiment is to investigate the activities of many people in different roles who work on a project concurrently. For that purpose, we have developed a project scenario, which is described in Section 7.2. The scenario gives an impression of the whole experiment and the interplay of different actor's roles. This is followed by the actual experiment, organized by role, and the experiment steps within each role organized chronologically.

We recognize that doing full planning for this small example might be overkill; nevertheless, we ask the experimenter to apply all the planning tools which the environment provides, even if the amount of work is small. We believe this is a less serious problem than the alternatives, which are to scale up the organization, plan, and scenario accordingly; or, to allow the experimenter to omit those steps or execute the steps informally, using pencil-and-paper methods. We have, however, tried to give the experimenter leeway to bypass some of the project mechanisms and to allow a variety of team organizations in the scenario.

5. Identification of Key Project Management (PM) Activities

In this section, activities within the specific area under investigation—in this case, project management—are identified, categorized, and refined. In this report, we do not classify the activities into primary and secondary functions. We are delaying the assignment of priorities until after we have applied the PM experiment to two environments.

5.1. Categories of Project Management Activities

Project management activities fall into four categories:

1. **Project Plan Management** represents the activities of a project manager during project planning, monitoring, and control.
2. **Plan Instantiation** represents the activities involved in reflecting project plans in the development support facilities of the environment, such that the project can be executed according to plan and progress can be reported to the project plan management facility.
3. **Project Execution** represents the activities of project members as they carry out the project simultaneously in a coordinated manner. The activities identified here relate to managing the development (task management, communication, coordination, etc.). Additional activities (e.g., design and coding) are included in the tasks of some members, but these are covered by other experiments in the evaluation methodology.
4. **Product Management** represents activities related to managing information and deliverables throughout the lifetime of the project.

   In general, product management is viewed as a major category of activities in software engineering, separate from project management. It includes product release control, change management, quality assurance, and configuration management (CM). Since release control and configuration management are already in place in the CM experiment, we decided to include the remaining product management activities as one of the four project management subareas at this time.

5.2. Refinement of Project Management Activities

For each major category of project management activities, we defined a set of specific user activities. We then arranged those activities into groups. Hints about how to instantiate an activity on a particular environment are shown in italics.

5.2.1. Project Plan Management (PPM)

Project Plan Management activities are organized into three groups:

1. **Project Plan Creation** represents activities related to the creation of initial plans. Once created, these plans provide input for setting up the development support facilities described in Plan Instantiation.
2. **Project Monitoring** represents activities related to monitoring and controlling the progress of projects in terms of the plans.
3. **Project Plan Revision** deals with adjustments to plans to reflect changes in the project. Such changes can range from regrouping teams to restructuring the product or the project.

Activities

1. **Project Plan Creation**

   a. Tailor planning support facilities.
   This includes: setting up the project calendar and specifying report formats, task description templates, and other project-specific templates; setting up default values for planning parameters, such as types of resources and their cost, and other cost estimation parameters as required by the cost estimation models.

   b. To test for traceability of project management documents, establish a relationship between the technical project objectives document (which is the basis for project planning) and the products, such as schedules.

   c. Develop a work breakdown structure (WBS) and work packages, including dependencies.

   d. Estimate the cost of work packages.
   This will be the raw data for the project cost estimation; includes pages of documentation, lines of code, man-weeks, and duration.

   e. Develop a project schedule.
   Includes critical path analysis and generating schedule report.

   f. Assign resources.
   Includes assignment of people, equipment, and other resources; handling personal time; resource utilization based on plan.

   g. Estimate the project cost.
   Run cost-estimation model(s). Functionality may include: compensation for communication overhead; handling man-months (time it takes to do a task) versus head count (actual number of people it takes); handling cost increases; cost assignment at different levels of detail.

   h. Merge team/group plans into global project plan.
   How well does the project planning facility support plan development in very large projects, i.e., plan development by team managers that feeds into plan development for the overall project?

   i. Generate a complete plan document.
   This is the document to be used as a baseline for carrying out the project.

2. **Project Monitoring**

   This set of activities represents project monitoring as it is done in terms of project plan management. It is assumed that information regarding progress is passed to the plan management facility in some form (addressed in Section 5.2.2, **Plan Instantiation**). Here the issues to be addressed are support for recording actual figures representing project progress, and support for trend analysis which compares actuals to plan figures.

a. Record and generate reports on actual progress data.
Determine what progress information can be recorded and tracked through the project planning facility. Includes schedule-related, resource-related, and cost-related information, and possibly comments/explanations about irregularities.

b. Analyze progress against schedule.

c. Analyze actual cost against estimates.

d. Analyze resource utilization.
Answer questions such as: Are the computers overloaded? Is disk space usage growing according to plan? May include reassigning people to unscheduled tasks, i.e, tracking ratio of activities that are not project-related.

e. Generate summary reports.
For example, trend analysis reports and periodic progress reports.

3. **Project Plan Revision**

This list of activities is not necessarily complete, but it does show the situations that commonly occur during the life of a project. There are two classes of events that cause plans to be changed: updates to reflect actual change (e.g., people leaving or equipment outage), and adjustments to keep a project on target according to plan. This requires what-if analysis to determine the effects of various alternative situations.

a. Baseline the plans.
Keep a record of all versions of plans.

b. Perform what-if analysis.
A generic activity; instances of particular planning activities with what-if analysis occur throughout the remainder of this list.

c. Handle schedule slippages by making schedule adjustments.

d. Handle adjustments in working hours.
For example, have team members work 48 hours a week instead of 40.

e. Cope with personnel changes.
Reassignments, general turn-over, vacation, illness.

f. Handle changes in WBS/task structure.
New tasks being introduced, tasks being merged.

g. Handle changes to the project structure.
Splitting and merging subprojects.

h. Handle changes in product deliverables.
Splitting and merging subsystems, changes in existing deliverables. Implies cost changes.

i. Adjust cost parameters based on actual data.
Personnel cost, computer usage.

j. Handle computing resource changes.
Computer outage, addition of new equipment.

k. Generate reports highlighting changes to plan.

5.2.2. Plan Instantiation (PI)

This set of activities deals with instantiating a plan in the development environment so that the project can be executed according to plan. The activities are divided into three groups:

1. **Project Installation** deals with setting up the development support facilities to reflect the project plans.
2. **Reporting Mechanism Installation** consists of instrumenting the development support facilities for reporting relevant information back to the plan management facilities.
3. **Reflecting Modifications to Plan** is essential to cope with changes in an ongoing project.

In a fully integrated environment, plan installation should be automated, and installation of reporting mechanisms should be limited to specifying when and how much information is reported. For other environments, such support will be lacking, and installation will require a lot of manual work by the user.

Activities

1. **Project Installation**
   - a. Set up product structure.
     Set up place holders for expected products; set up default ownership and access rights; set up necessary information to reflect standards and formats.
   - b. Set up team structure.
     Reflect team hierarchy of plans; set up access control; assign equipment; set up work areas; set up user profiles.
   - c. Set up task structure.
     Reflect work packages and their dependencies; set up mechanisms for activating tasks based on schedule and dependencies; set up mechanisms to tie together products, people, and tasks.
2. **Reporting Mechanism Installation**

   This set of activities represents setting up reporting mechanisms to periodically report information to the project planning facilities. Progress reporting to the planning system should be happening in several areas: accounting information, task completion and product delivery, resource consumption to anticipate bottlenecks, etc.
   - a. Provisions for reporting task completion and delivered products.
     To validate estimates, document size may be relevant as well as actual time to complete the task.
   - b. Provisions for reporting accounting information to match up to budgets.
     Personnel time, resource usage.
   - c. Provisions for reporting statistical data.
     For example, compilation error statistics, which some managers use to discover problem spots.
3. **Reflecting Modifications to Plan**

   This set of activities should uncover in more detail how well the development support facilities cover concepts desirable from the viewpoint of project plan management, as well as how flexible the chosen support mechanisms are when making changes to the project. The following list of items has a close correlation to the list in Project Planning above.

a. Reassignment of people.
   Should raise issues of access control, ownership, etc.

b. Changes in task structure and schedule.

c. Changes in project structure and product structure.
   For example, splitting a subsystem into two with appropriate adjustments in product structure and task descriptions; merging two subprojects into one, possibly implying merging the underlying databases. Implications for access control.

d. Changes in assignment of computing resources.
   For example, move a subproject onto a different machine.

5.2.3. Project Execution (PX)

The execution of a project plan involves two related groups of activities:

1. **Communication and Coordination** addresses issues of the exchange of project information as it relates to the structure of the product, the team, and the task.
2. **Information Access and Control** examines issues of the transparency of the physical database organization, and of limiting access to information in the database.

Activities

1. **Communication and Coordination**
   a. Provide communication between team members.
   This includes the direction of communication, the type of information communicated, the communication paradigm, the role of the system in enforcing rules of exchange, the linkage between tasks and the communication of task information, the presence of automatic constraint checking on tasks and their orderings, and the performance of project communication tools.

   b. Coordination of work area.
   Controlled isolation of work area, environment support of work area, transparent access to environment database from work area, coordination of access to product structure.

   c. Task completion and notification.
   Querying status of teams, tasks, products; exit criteria; automatic versus manual notification; automatic propagation of changes; propagation of change notification (e.g., attaching notifications to related objects or to the task list of the responsible person).

2. **Information Access and Control**
   a. Project database access.
   Ability to partition database and provide transparent access.

   b. Access control.
   Types of access, physical versus logical entities, granularity of access control (e.g., spec/body at package or procedure level, "with" clause).

5.2.4. Product Management (PDM)

Product Management activities include: the generation and storage of project-related documents; methods or tools for tracking connectivity, enforcing traceability, checking conformance to standards, controlling product releases, and managing changes; and support for standards.

Activities

1. **Traceability of Project Documentation**
    a. Access trace information.
    For storing project information and documentation, what are the structures, and how are they accessed?

    b. Creation of trace information.
    Entering or establishing traceability to track the relationship of objects in the database (e.g., requirements to designs to code).

2. **Control of Change Requests/Error Reports**
    a. Logging, tracking, status queries.

    b. Approval of requests.
    For example, can only authorized persons issue or approve change requests?

    c. Support for relating change requests and error reports to each other and to objects in product structure.
    For example, can change requests be attached to releases? Is the mapping of error reports to change requests one-to-one, many-to-one, one-to-many, or many-to-many?

3. **Quality Control**
    a. Checking adherence to standards.

    b. Validation and acceptance testing.

    c. Support for test plan development.
    What standards are supported?

4. **Product Release Control**

    (This activity is covered in the Configuration Management experiment.)

5.3. Activity Codes

In Section 7.3, we cross-reference these project management activities using a numbering system composed of the initials of a major category, followed by the group number, followed by the activity letter. For example, activity PDM-3.a refers to the major category Product Management, group 3, activity a: Checking adherence to standards, listed on this page.

6. Definition of Evaluative Criteria and Questions

This section establishes the specific criteria to be used in the evaluation and documents a list of evaluation questions covering each criteria area.

6.1. Evaluative Criteria

The evaluative criteria fall into four categories: functionality, performance, user interface, and system interface. The criteria are summarized below in tabular form. The evaluative criteria are discussed in detail in [2].

- Functionality
  - Completeness in major functional areas.
  - Level of support for activities.
- Performance
  - Execution time efficiency.
  - Space efficiency.
  - Responsiveness.
- User Interface
  - Ability to be learned/used.
  - Consistency/uniformity.
  - Helpfulness.
  - Error handling.
  - Communication.
- System Interface
  - Ability to be customized.
  - Integration with other environment tools.
  - Openness to new tools.
  - Portability.
  - Utilization of operating system.

6.2. Evaluative Questions

The questions listed in this section represent the core of the evaluation. The questions are grouped according to the four criteria previously outlined.

With respect to the **Functionality** criterion, the experimenter should describe the mapping of each generic experiment step onto the instance for a target environment. This description is supported by the command sequences in terms of the target environment. Thus, the functionality questions do not ask the experimenter to describe a particular experiment step. In the question/answer session, the experimenter fills out the functionality checklist that has been provided. Each entry can be annotated if desired (e.g., with comments regarding the particular plan representations supported such as PERT, GANTT, CPM). The functionality checklist is open-ended in that the experimenter can point out activities that are part of the target environment but are not covered by the given functionality list. In answering the questions for each of the sub-areas within project management, the experimenter provides insight as to how well the target environment supports the project management activities, and what the architectural similarities and differences are between target environments.

The **Performance** criteria section lists questions for which the experimenter should perform accurate time and space measurements. In addition, a set of responsiveness questions asks the experimenter to comment on whether certain activities provide acceptable response time based on approximate measures. By "acceptable" we mean that activities which appear trivial or are performed frequently should execute quickly (in seconds), whereas other activities can take longer (minutes, perhaps).

The **User Interface** criteria section asks questions that can be answered for the complete environment, assuming that the user interface is reasonably uniform across all parts of the environment. For parts of the environment that are not integrated, the tools will probably present a different user interface, and the experimenter should comment on them in addition to giving general answers.

The **System Interface** criteria section probes issues of the integration of tools (both within project management and between project management and other areas of the environment), the openness of the environment to new tools or additional functionality, whether the environment can be customized or tailored to the user (via user profiles, for example), the portability or availability of the environment on diverse hardware, and the mapping from the environment to the underlying operating system (if there is one) which includes issues of access control and concurrent access to the development database.

6.2.1. Functionality

1. **General questions**

   a. Fill out the functionality checklist for each of the four subareas of project management. Which project management activities are supported, and which ones are not?

   b. How well does the environment cover the management of all deliverables, plans, and products?

   c. To what degree does the environment impose a management style or management policies?

   d. How well can the environment be adapted to a particular organization?

   e. To what degree can the environment support distributed project development?

   f. Does the environment encourage or support reusability in a formal way? Is there a library of reusable software components? What can be placed in it—plans, code, designs, etc.—and how is it searched/accessed?

2. **Project Plan Management questions**

   a. What cost estimation models are used? Are they sufficiently flexible to be tailored to reflect the characteristics of the organization or project?

   b. Are the information structures for project plan management sufficiently rich and extensible to accommodate the information needs (e.g., are there enough placeholders for relevant information including comments/annotations to be included in the plan information)? Are the structures integrated, or does the user need to duplicate information?

   c. How well is checking for inconsistencies and constraints in plans handled (e.g., overassignment of resources, budget overruns, critical path)? For example, if a person is reassigned, is all information about that person's work on the project updated correctly?

   d. What are the supported reporting formats for project plans and progress information (e.g., PERT, GANTT, trend charts, resource charts)?

   e. How well does the project planning facility support what-if analysis?

   f. How much support is there for synchronizing plan development by multiple people? for merging plans?

   g. How well does the project planning facility support both planning-in-the-large and planning-in-the-small? (Planning-in-the-large refers to activities such as global cost estimation and global resource assignment [e.g., number of people]; planning-in-the-small represents activities such as assignment of individuals.)

   h. How flexible is the project planning facility in handling different team structures? Does it support certain team structures better than others?

3. **Plan Instantiation questions**

   a. How (and how closely) can the planning information be reflected in the development support facility to guide the development? Is there support for developers to track and manage their tasks and to work in the context of a task?

b. How automated is the support for setting up and maintaining development support facilities to reflect current plans? Is the project plan tied to development such that it must reflect the current status of the project?

c. How well is the development facility insulated from what-if analyses of planning activities? How difficult is it to merge a new plan with ongoing project execution?

d. How adequate is the support for frequent changes in a project during its lifetime?

e. How automated are the facilities for reporting project monitoring information to the planning and monitoring facility?

f. Is there a conflict of interest between setting up accounting structures and access control structures? (With the UNIX operating system, for example, disk usage accounting as well as access control is based on ownership. Disk usage on a per team basis can be done on the basis on group ownership; this may conflict with one team sharing software with the other team but not with other users.)

g. What statistics can be collected by the environment (e.g., bugs per subsystem)?

4. **Project Execution questions**

a. What means exist for finding out the status of teams, tasks, and products (e.g., on-line queries, interim or periodic reports)? What type of queries are supported? How easy is it to add custom queries?

b. If task lists are supported, are they strictly private to each user, or is their information shared among team members (e.g., passed down a hierarchy)? Are tasks assigned to physical persons or to logical roles? (Different people can assume the same role, such as maintainer of a library).

c. What does the environment support for logical groups and accounts? (Can access rights be mapped to a task or only to a person? Does the environment have the notion of a "task description" which automatically links a task with the users assigned to it?)

d. Can project information and status be communicated horizontally (between peers), vertically (between supervisor and subordinate), or both? What is the communication paradigm (point-to-point like electronic mail, or broadcast like bboard)?

e. When information is communicated about a project’s tasks, status, or resources, what is the information content? (Is the task description passed to the user text interpreted only by the reader, or structured information including schedule information or design fragments?)

f. How "involved" is the system in this communication? Are there protocols to assist the exchange of project information or enforce rules of exchange, i.e., are specialized information flow patterns supported?

g. How automated is the support for task notification and completion? For example, does the system do automatic checking of constraints on tasks and their orderings? If one task is dependent on the results of another task, how is it activated? What happens when a task is completed; are team members automatically notified? Does the cascading of change requests or task completion messages create a "ripple effect"?

h. Does the environment have a means of grouping tools, e.g., to support different roles? Is the contents of a group fixed by the system or under user control? Can the functionality of the environment be divided into subsets so as to tailor it to the needs of the individual user?

i. Can teams intersect, i.e., can one person on a project simultaneously be a member of more than one team?

j. How does the environment support the user in the user's workspace or working directory? (After having software checked out is the user interacting with the environment or the regular operating system and command interpreter while programming? Does the environment help the user by managing error messages, by providing transparency between the envrionment repository and workspace?)

5. **Product Management questions**

a. How does the system track connectivity? How does it enforce traceability to requirements? What is the interrelationship of documents (as opposed to code)? Are pointers kept that associate, for example, an Ada module with its design or its test plan? How are relationships represented (e.g., as pointers in text, as relationships about objects)?

b. What tools exist for generating standard deliverables and documents (e.g., MIL-STD-2167)? for deriving documents from other documents?

c. What mechanisms exist for managing and controlling change requests? (Change requests include requests for bug fixes, requests for added functionality, improvements in the user interface, and performance enhancements.)

d. How are software bugs reported and tracked? Can constraints be imposed regarding who can submit reports and what reports can be submitted against? Can bug reports be related to change requests?

e. How is adherence to standards and procedures checked?

f. Are the deliverables from a software project required to undergo formal acceptance testing?

g. What assistance does the environment give the user to evaluate the quality of deliverables (path testing, code audits, Q/A plans, etc.)? Are there tools for rating/ranking quality factors?

h. Are the formal quality standards for a project kept on-line?

i. What mechanisms exist for communicating or reporting between Q/A and the developers?

j. Is the user's workspace insulated from changes in the developed product? (For example, if a user has reserved components in an existing library, and a new version of the library is installed, which version will be picked up?)

6.2.2. Performance

1. What is the elapsed time for the following project management activities:
    - a. instantiating a plan, i.e., setting the development support facility up to reflect the project plan *(to be measured only if not manually performed)*
    - b. generating a plan document and status reports
    - c. standards checking (e.g., coding standard) if provided
    - d. retrieving related documents (using traceability relations)
    - e. processing progress data (for trend analysis)
    - f. opening/closing a work area
    - g. creating a task
    - h. notifying project members (with full propagation) of task completion
    - i. executing a status query, e.g., modules to be compiled
    - j. making an object not actively being worked on accessible in a work area for reading (if not already accessible transparently).
2. What is the storage cost of the following? Consider both the fixed, or one-time, cost to set up the facility, and the marginal cost of storing one additional message, plan, etc.
    - a. plans (schedule, WBS, resource management)
    - b. plan instances (product structure, team structure, task structure)
    - c. progress information
    - d. change control information
    - e. bug reports and change requests
    - f. work area overhead (i.e., an empty work area)
    - g. plan alternatives (from what-if analysis)
    - h. project statistics
    - i. relationships of objects (i.e., cost of tracability).
3. What is the responsiveness of the project management facility for each of the following:
    - a. plan development, monitoring, and revision
    - b. reporting information from development facilities to planning facilities
    - c. communication between project members (i.e., what time it takes for the recipient to become aware of a message after the sender submitted it to the communication facility such as electronic mail or environment specific mechanisms)

d. change management
e. access control
f. critical path analysis
g. context switching between plan alternatives (during what-if analysis)
h. collection of project statistics.

6.2.3. User Interface

For definitions of terms used in this section, refer to [4] or [6].

1. **Usability/Learnability**
    a. How easy/difficult is it to learn the project management facilities? (For example, is the command syntax awkward, mnemonic? Is command completion provided? Does the system offer selection from legal alternatives?)
    b. How easy/difficult is it to use the project management facilities being a knowledgeable user? (For example, is there an efficient interaction mode? Can menu and forms prompting be disabled?)
2. **Consistency/Uniformity**
    a. How consistent and uniform is the user dialogue (i.e., the command syntax, use of menus, etc.)?
    b. How consistent and uniform are the naming conventions, on-line help facilities, error diagnostics and handling?
    c. What degree of user customization is supported (e.g., change key bindings, write your own command procedures)?
3. **Helpfulness**
    a. How much of routine interaction is streamlined through automation? (Examples are: providing command completion or last-used name as default for parameters on the user interface level, and composite operations to automate steps with possible confirmation by the user on the action/command level.)
    b. How much context sensitive on-line assistance is provided?
    c. How complete, concise, and appropriate is the documentation?
4. **Error Handling**
    a. How much tolerance does the environment show for minor errors (e.g., syntax errors)?
    b. How does the environment cope with mistaken use of commands that have potentially disastrous results (e.g., by requesting confirmation or by providing an *undo* facility)?
    c. What is the quality of the error diagnostics (early and correct detection, appropriate identification and description, differences in on-line/interactive and printed/batch run diagnostic messages, brief or full error reporting)?
5. **Communication**
    a. How well does the environment use the available hardware for communication with the user (e.g., pointing devices, multi-window multi-font screens)?
    b. Is the quality of the information presentation acceptable (e.g., legibility and size of fonts, choice of background color, placement of windows and menus, key bindings)?
    c. What is the degree of support for multiple views of information (this includes formatting, elision, and browsing)?

d. Is the degree of interactiveness/responsiveness acceptable (e.g., are diagnostic reports timely; are "simple" functions inexpensive to perform)?

6.2.4. System Interface

1. How tailorable are the report generation facilities for project plans and status reports?
2. How tailorable is the environment to project- or company-specific document formats?
3. How is the team structure and access control accomplished (mapped onto an underlying operating system)?
4. How are the various objects in project management (tasks, plans, teams, products, etc.) mapped into an underlying storage structure (e.g., file system, database)? Is this information stored and accessed in a central repository, or in isolated files managed by stand alone tools?
5. Is the environment cognizant of changes in the outside environment? (For example, how do the environment and the workspaces know when a new Ada compiler is installed?)
6. How good is the coordination and synchronization of the development database (issues such as locking, query while update)?
7. If the environment is built on top of an operating system, how efficient is the interface? For example, does the environment use the operating system's protection mechanism or duplicate the lower-level mechanisms? Or, if a communication tool such as electronic mail is layered on top of the system, is it possible to subvert the environment's facilities and deal with lower-level access primitives? Is there an incentive to do so, e.g., better performance, easier to use?
8. How portable is the project management environment with respect to hardware? with respect to different operating systems?
9. How integrated are the project planning facilities with the rest of the environment?
10. How well are the different planning tools integrated within the project planning facility?
11. Are there well-defined data-exchange formats to pass project planning information to external project management tools?
12. Is it possible to apply the environment to an existing project (i.e., to import project-related data which already exists)? How much effort is required to do so?

6.3. Question Codes

In Section 7.3, we cross-reference these questions using a numbering system consisting of the first letter of the criteria (F, P, UI, SI), followed by the number of the question category, followed by a lower-case letter indicating the specific question(s). For example, question UI3c refers to the User Interface criteria, category 3, part c: How complete, concise, and appropriate is the documentation?

7. Phase 3: The Generic PM Experiment

The PM experiment description is organized as follows. First, a description of the experiment context and setup is given, followed by a short description of the experiment scenario. The main part, Section 7.3, describes the experiment steps in detail. Each step consists of a description, a list of activities and questions that are covered by the step, and hints to the person instantiating the experiment for a particular environment.

7.1. The Context

A software system has been produced and released. The product consists of several subsystems, one of which is the user interface (UI) subsystem described and used in the configuration management (CM) experiment of the Evaluation of Ada Environments methodology. The UI subsystem is illustrated in Figure 7-1.

Figure 7-1:   Software System Structure

The setting for the PM experiment is an organization that is responsible for the maintenance and enhancement of the delivered product. The organization has available information from the original development and a pool of resources that includes people with different degrees of knowledge of the product domain and product. The organization is split into two groups: a maintenance group and an enhancement group. In the PM experiment, we concentrate on the maintenance group. The enhancement group may come into play in future extensions of the experiment to investigate support for more complex multiple team interactions.

The organization is fictitious but representative; it is illustrated in Figure 7-2. We are attempting to examine the environment for support of different organizational structures.

Figure 7-2: Organizational Structure

The maintenance group of the UI subsystem is organized as follows:

- a manager
- a system analyst
- five teams:
    - three maintenance teams (one for each part of the subsystem)
    - a documentation group
    - an integration and quality assurance group

The manager is responsible for the overall planning and management of the maintenance activities. Some activities that we place under the manager may be those of an administrative assistant, but we refrained from splitting them out for the sake of simplicity. The system analyst is responsible for determining necessary maintenance changes.

The maintenance teams make the actual changes and perform unit testing. Team 1 is structured as a team leader and two team members, with the team leader responsible for the team's plans and for integrating and releasing their deliverable. The team members are restricted in their interactions with other teams in that certain activities require approval of the team leader. Team 2 is structured as a set of two cooperating but independent members. From the outside they are viewed as one entity, and tasks are assigned to the team, not individual members. Team 3 consists of a single member, i.e., we are testing the environment's ability to minimize management overhead. The documentation group is responsible for maintaining the user documentation, which is part of the product release. The integration-and-quality assurance group is responsible for working out a quality assurance (QA) plan, performing acceptance testing and integration of team deliverables, and packaging of the new release. We assume that each of these two groups consists of one person.

7.2. The Scenario

In this section, the scenario around which we have organized the PM experiment is described. An overall graphical view of the scenario may be found in Appendix E.

Two customers have filed a number of error reports (see Figure 7-3). We have chosen the collection of error reports and their implications in such a way that it allows the experimenter to examine how well an environment supports the management of such reports. The set of error reports will exercise tracking of error reports, mapping of error reports to fixes, relating error reports to each other, and relating responses to error reports.

Figure 7-3: Error Reports and Respective Actions

The manager initiates maintenance activities which will culminate in a new release. The manager requests that QA refine a plan for the next release and sends the error reports to the system analyst for analysis. The system analyst classifies the error reports, traces the development history to locate the cause of each error, determines the scope of corrective changes by querying dependency information, and specifies what actions are necessary. QA develops a quality assurance plan as well as standard forms and checklists, and installs form support; this is illustrated in Figure 7-4. This scenario allows the experimenter to examine informal task assignment and tracking support or, alternatively, to assess the overhead of project planning required for assigning a small set of tasks.

The manager makes an initial global plan, illustrated in Figure 7-5. This plan is passed to team leaders for refinement. It includes task descriptions to implement bug fixes based on the change recommendations from the systems analyst. The team leaders report back their refinements, which are then merged into one plan by the manager. The plans are approved by the manager and instantiated. The parallel plan refinement and merging of plans is shown in the second part of Figure 7-4. This scenario gives the experimenter a chance to investigate support for concurrent planning by multiple people. It also investigates the restrictions imposed by planning tools on management styles. For example, a tool that requires assigning a task to individuals rather than to a group of people (or logical entity) does not allow the members of Team 2 to choose among assignments.

Figure 7-4: Initial Activities

Figure 7-5: Initial Global Plan

The plan for Team 1 is illustrated in Figure 7-6. Team 1 has changes requiring an update to documentation. A design review is scheduled for all members. This investigates the ability of the planning tool to express a whole team as a resource, as well as the ease of assigning one person to another task in the middle of a longer task. After the design review, the documentation group starts working on an update to the user manual. This tests for the ability of the environment to automatically activate a dependent task. Programmer T12 leaves in the middle of his task and is replaced by T12suc. Because the new programmer requires a week to get familiar with the task, and the task is on the critical path, the schedule would slip. An investigation of plan alternatives should come to the conclusion that no slippage is necessary if the new programmer does not participate in the design review. This scenario tests how visible the slippages in the critical path are, and how well the project planning copes with plan changes and conflicts. These changes to the plan are illustrated in Figure 7-7.

Figure 7-6: T1 Plan

Team 2 is assigned several tasks. Its members select a task themselves from the assigned tasks; this represents a more cooperative setting. During the task of one member, the corrections in the code potentially require a change in another part of the UI subsystem. By attempting to make this change himself, the member tests the ability of the environment to control change.

Figure 7-7: Modifications to T1 Plan

Figure 7-8: Integration Plan

QA accepts deliverables from three teams and merges the new versions into an integrated subsystem. QA receives documentation, then completes a release document. The integration process is illustrated in Figure 7-8. The manager signs off on the new release, and it is distributed to appropriate customers.

Figures 7-9, 7-10, and 7-11 illustrate how the source code of subsystem UI and its components evolves over time. The upper part of each figure shows who is involved in creating the new version of the respective component. In the lower part of each figure, the version numbers of the configurations of the UI, command language interpreter (CLI), and screen management (SM) are listed on the left. For each version of the configuration, the thread selecting the appropriate version of each configuration component is shown by a line of a unique pattern starting from the version label. The version numbers in the boxes on the right of Figure 7-9 indicate the versions of the respective parts CLI, SM, and virtual terminal (VT). Finally, Figure 7-12 illustrates the package that is delivered to the customers.

Figure 7-9: Version History of UI Subsystem

Figure 7-10: Version History of CLI

Figure 7-11: Version History of SM

Figure 7-12: Customer Deliverable

7.3. The Experiment

The experiment consists of concurrent activities by multiple people. Because it is difficult to describe concurrent activities in sequential form as experiment steps, we have organized the experiment steps into groups that represent the roles of different people, such as customer, manager, and documentation group. The interaction between the roles should be identifiable from the scenario description above and from the passing of deliverables or information. Within each role, the experiment steps are described in temporal order.

See Sections 5.3 and 6.3 for an explanation of activity and question codes.

7.3.1. The Experiment Setup

The steps listed here must be done before the actual experiment can be carried out. Setup includes setting up the environment's development database to contain the initial release of the system, as well as tailoring the environment to a specific organization or project.

> *In addition, the person instantiating the experiment on a particular environment will have to determine the appropriate mechanisms for collecting timing and size information to answer performance questions.*

First, set up the development database. Then initialize the environment with project-specific parameters regarding this experiment.

1. Load the source code for the UI system and record it in the development database. (Version histories and configuration threads for UI, CLI, and SM are diagramed in figures 7-9, 7-10, and 7-11.) If the CM experiment has been completed, use its source code configuration of UI as release 1.0.

   PERFORMANCE MEASUREMENT: Record the storage cost for UI source.

   ACTIVITIES: PI-1.a

   QUESTIONS: P2b

2. Create a design document for each of the three subsystems of UI[2] and enter them into the development database as versions. The purpose is to demonstrate the ability to relate and trace documents, which, in some environments, requires placing pointers in the document's content (the text, for example).

   PERFORMANCE MEASUREMENT: Record the storage cost for the design and for traceability relations.

   ACTIVITIES: PI-1.a

   QUESTIONS: P2b, P2i

3. Relate design documents (their final versions) to release 1.0 of the UI source code to represent traceability.

   ACTIVITIES: PI-1.a, PDM-1b

4. Create a user manual document, version 1.0, for UI.[3] Enter it into the development database, and relate it to release 1.0 of UI.

   PERFORMANCE MEASUREMENT: Record the storage cost for the user manual.

   ACTIVITIES: PI-1.a

---

[2]Content is irrelevant.

[3]Content is irrelevant for this experiment, other than a reference to CLI.

QUESTIONS: P2b

5. Package up the executable code and the user manual as a customer deliverable, release 1.0. (See Figure 7-12.)

   PERFORMANCE MEASUREMENT: Record the storage cost for the deliverable.

   ACTIVITIES: PI-1.a

   QUESTIONS: P2b

6. Initialize calendar with work hours, work days, holidays.

   ACTIVITIES: PPM-1.a

7. Enter persons as available resources for the project. Different individuals have different qualifications (analysis, documentation, management, etc.); see Figure 7-2 for details. Enter planned vacation for documentation person during second week after detailed plan has been approved.

   ACTIVITIES: PPM-1.a

8. Carry out system administrative initialization such as default printers, report formats. Make use of whatever support the environment offers in grouping tools or creating logical subsets of the environment for specific users.

   PERFORMANCE MEASUREMENT: If work areas are set up at system initialization time, record elapsed time and space to create a logical work area for a member of one team.

   ACTIVITIES: PPM-1.a, PI-2.a, PI-2.b, PI-2.c

   QUESTIONS: F4h, P1f, P2f

7.3.2. The Customers

The two customers, CU1 and CU2, are using UI Release 1.0; they encounter problems, and submit error reports. They are informed of the treatment of these reports, and will receive the new release with the expected bug fixes.

1. Customer CU1 submits four error reports regarding UI release 1.0 to the UI customer service address CS.UI@<Company>. (The actual text of the error reports is not relevant for the purpose of this experiment, unless the environment provides special features for content processing that should be highlighted as part of the evaluation.)

2. Customer CU2 submits four error reports regarding UI release 1.0.

3. Customers receive and examine responses regarding treatment of the submitted reports.

4. Customers receive a release notice for UI release 1.1, which indicates changes in the new system. They try to relate the information in this document to the submitted error reports and earlier responses. Customers request actual delivery of release 1.1.

5. Customer CU2 checks on status of his first error report (report #5). The initial response had stated that it would be handled in an enhancement release. Customers receive delivery of UI release 1.1.

   ACTIVITIES: PDM-2.b

   QUESTIONS: F5e

7.3.3. The Manager for Product Maintenance

This individual is responsible for handling error reports received by customer service.

1. Generate a report of error reports (on-line and hardcopy).

   ACTIVITIES: PDM-2.b

2. Initiate error report analysis task for system analyst and request a response within five days.

3. Initiate task to QA to adjust Q/A plans for maintenance release (release 1.1) and to define a release note document format.

   PERFORMANCE MEASUREMENT: To test the minimal overhead due to planning activities, consider the trade-offs, on the one hand, of carrying out a planning step with resource allocation or, on the other hand, of assuming that the manager does informal resource allocation negotiation with QA and the system analyst.

4. Receive recommendations from system analyst, respond to customer about report #5 by recommending its accommodation in the next enhancement release (release 2.0), and inform manager of enhancement project. Approve recommended change requests.

   PERFORMANCE MEASUREMENT: Note responsiveness of change management facilities.

   ACTIVITIES: PDM-2.a

   QUESTIONS: P3d

5. Turn remaining recommendations into an initial global plan (see Figure 7-5).

   a. Define work packages in the initial WBS for three maintenance teams, a documentation group, and a Q/A group.

   b. Estimate man-days, number of resources (persons), number of days, number of changed lines of code.

   c. Perform a cost estimation and set up a budget for the project as well as for teams.

   d. Work out an initial global schedule.

   e. Generate a document containing the initial global plan. If possible, generate different views of the plan information, e.g., PERT chart, work package listing, resource chart.

   f. Retain a version of the plan as part of the project history.

   PERFORMANCE MEASUREMENT: For each sub-step above, record the responsiveness of the tool or facility used in plan development. Record the storage cost of each object in the global plan: WBS, schedule, PERT chart, cost estimate, etc. Note how long a critical path analysis takes as an indication of interactiveness.

   ACTIVITIES: PPM-1.b, PPM-1.c, PPM-1.d, PPM-1.e, PPM-1.g, PPM-1.i

   QUESTIONS: F2a, F2b, P2a, P3a, P3f, S4

6. Issue tasks to the documentation group, the Q/A group, and the three maintenance teams for plan refinement and feedback. Teams 2 and 3 and Documentation are requested to confirm their aspect of the plan. Team 1 and Q/A are requested to refine their part of the plan.

   ACTIVITIES: PI-1.c

7. Merge refined plans from teams into a global plan. Perform consistency checks on

the new version of the plan: budget overrun, schedule overrun, overassignment of people, etc.

ACTIVITIES: PPM-1.h

QUESTIONS: F2c, F2f, F2g

8. Save the new version of the plan as project history. Generate a document containing the plan. If possible, generate a report highlighting changes in the two versions of the plan.

   ACTIVITIES: PPM-1.i

   QUESTIONS: F2d

9. Approve the plan. Inform customers of release schedule for error reports being handled by UI release 1.1. Inform teams to proceed according to approved plan, e.g., by issuing tasks. Set up access control so that only the team responsible for a system part has "modify" access rights, while others have only read access to the specification. (Modifying the specification of a part requires manager approval.) In the case of Team 1, inform the team leader, who in turn will issue the tasks to the team members.

   PERFORMANCE MEASUREMENT: Record elapsed time for plan instantiation. Record storage cost of plan instances. If work areas are set up when tasks are issued, record elapsed time and space to create a logical work area for a member of one team.

   ACTIVITIES: PPM-1.f, PI-1.b, PI-1.c

   QUESTIONS: F2h, F3a, F3b, F3d, F3f, F4i, P1a, P2b, P2f

10. Generate first monthly progress report. Produce summary report as well as complete report of all views supported by the project management software. Record progress report in project history.

    PERFORMANCE MEASUREMENT: Record elapsed time and storage cost to produce reports and get project statistics.

    ACTIVITIES: PPM-2.a, PPM-2.b, PPM-2.c, PPM-2.d, PPM-2.e

    QUESTIONS: P1b, P2c, P2h, P3h

11. Receive notice that personnel change in Team 1 causes tasks on critical path to slip. In the process of what-if analysis, query the status of the project (teams, tasks). Generate a report highlighting the plan changes. Determine that no slippage is necessary if new team member, T12suc, does not participate in design review. (And consider how the plan would be changed if it were necessary to increase the number of working hours or add staff to the project.) Inform leader of Team 1 to reflect this fact in his plan execution. Record the plan revision in the project history.

    PERFORMANCE MEASUREMENT: Record elapsed time for status query.

    ACTIVITIES: PPM-3.a, PPM-3.b, PPM-3.c, PPM-3.d, PPM-3.e, PPM-3.k, PI-3.a, PX-1.a

    QUESTIONS: F2e, F3b, F3c, F3e, F4a, P1i

12. Consider other changes in the project. These include changes in work breakdown or task structure, changes in schedule, changes in project structure, changes in product deliverables, adjustments to cost parameters based on actual data, and changes in computing resources.

    PERFORMANCE MEASUREMENT: Record storage cost of plan alternatives. Note responsiveness of system when context switching between alternatives.

    ACTIVITIES: PPM-3.f, PPM-3.g, PPM-3.h, PPM-3.i, PPM-3.j, PI-3.b, PI-3.c, PI-3.d

    QUESTIONS: F3d, P2g, P3g

13. Generate second monthly progress report. Perform a trend analysis which should show that delivery of a design document to the Documentation group slipped, but that the schedule is not affected. Record progress report in project history.

    PERFORMANCE MEASUREMENT: Record elapsed time to process progress data for trend analysis.

    ACTIVITIES: PPM-2.a, PPM-2.b, PPM-2.c, PPM-2.d, PPM-2.e

    QUESTIONS: F2b, F4a, F4e, P1e

14. Receive customer release from Q/A and approve it for release. Generate a report with statistics on project execution (e.g., computer utilization or lines of code changed) to mirror the organization's concern for metrics.

    PERFORMANCE MEASUREMENT: Note responsiveness of communication.

    QUESTIONS: F3g, F4e, F5f, P3b

15. Generate distribution list based on filed error reports that have been handled by this release. Distribute release notice to customers.

    ACTIVITIES: PDM-2.b, PDM-2.c, PDM-4

16. Invite all project members to a release completion party.

    ACTIVITIES: PX-1a

17. Distribute customer delivery release 1.1 to responding customers.

    ACTIVITIES: PDM-4

7.3.4. The System Analyst

The system analyst receives the collection of error reports, analyzes them and the source code of the corresponding release, and provides recommended actions to the manager for each problem report.

1. For error report #1, prepare and send a response to customer indicating that the reported item is not an error in the UI system.

   ACTIVITIES: PDM-2.a, PDM-2.b

2. For error report #2, indicate a recommended change to VT with an indication of the complexity of the change.

3. For error report #3, indicate a recommended change to CLI package **str_utilities** with an indication of the complexity of the change.

4. For error report #4, examine the user manual, trace to the source code of CLI, examine its related design document, and recommend a change to the design, source code, and user manual. The affected package is **command_interpreter**.

   PERFORMANCE MEASUREMENT: Record elapsed time to use the traceability relations.

   ACTIVITIES: PDM-1.a

   QUESTIONS: F4j, F5a, P1d

5. For error report #5, indicate that the reported item should be handled as an en-hancement.

6. For error report #6, indicate a recommended change to SM package **window_manager** with an indication of the complexity of the change.

7. For error report #7, indicate a recommended change to SM package **viewport_manager** with an indication of the complexity of the change.

8. For error report #8, indicate that the reported item is the same as reported in #2.

9. Pass recommendations for error reports to manager (if possible, in the form of proposed change requests).

   PERFORMANCE MEASUREMENT: Record storage cost of change control information.

   ACTIVITIES: PX-1.a, PX-1.c, PDM-2c

   QUESTIONS: F4g, F5e, P2d

7.3.5. Team 1

This team has three members: a leader and two programmers. It is responsible for the packages in UI_CLI.

1. Team leader receives global plan from manager, refines it and assigns people to tasks, and sends a detailed plan back to manager.

   PERFORMANCE MEASUREMENT: Record elapsed time to create a task. If work areas are set up by programmers, record elapsed time and space to create a logical work area for one programmer.

   ACTIVITIES: PPM-1.f, PX-1.a

   QUESTIONS: F4d, F4e, F4i, P1g, P2f

2. Team leader accepts tasks **3** and **4** from manager, sends **4** to programmer #1 and **3** to programmer #2.

   PERFORMANCE MEASUREMENT: Note responsiveness of communication.

   ACTIVITIES: PX-1.a

   QUESTIONS: F4b, F4d, F4e, P3c

3. Programmer #1 reviews bug report in **4** and begins design change. Programmer #2 begins change to code.

   PERFORMANCE MEASUREMENT: Record the elapsed time to bring modules into work area.

   QUESTIONS: F5e, P1j

4. Leader reports progress and resource consumption at end of week 2.

   PERFORMANCE MEASUREMENT: Record elapsed time to create report.

   ACTIVITIES: PX-1.c

   QUESTIONS: F3e, F4d, F4e, P1b, S1

5. Programmer #2 quits the team in week 3 of his assignment. Team leader requests manager to assign a new employee to that task, proposes an adjusted plan for team 1 (still assuming all members participating in the review) and gives estimate of slippage caused by replacement of personnel. Receives approval from manager.

   PERFORMANCE MEASUREMENT: Note responsiveness of change procedures.

   ACTIVITIES: PI-3.a, PX-1.a, PX-2.a, PX-2.b

   QUESTIONS: F3d, F3e, P3d

6. Team leader grants the new programmer (T12suc) the same access to the source, designs, etc. that the ex-programmer (T12) had.

   PERFORMANCE MEASUREMENT: Note responsiveness of access control.

   QUESTIONS: F3d, F4c, P3e

7. Programmer #1 finishes design change (as approved by team).

8. Design review by entire team.

9. Team approves design change and sends document to Documentation group.

   ACTIVITIES: PX-1.a

   QUESTIONS: F4e, F4g, F5e

10. Programmer #1 makes change to code and tests, then passes new version to leader.
11. New programmer #2 completes change to code and tests, then passes new version to leader.
12. Leader reports progress and resource consumption at end of week 6.
13. Leader integrates changes, tests, and notifies the two programmers of successful test. Approves new release of UI_CLI and sends it to Q/A. Notifies manager.

    PERFORMANCE MEASUREMENT: Record total storage space of messages for team leader, programmer #1, programmer #2. Record elapsed time for notification of task completion.

    ACTIVITIES: PX-1.a, PX-1.c

    QUESTIONS: F4c, F4d, F4g, F5e, P1h, P2e

7.3.6. Team 2

This team has two members who work semi-independently. They are responsible for the packages in UI_SM.

1. Receives global plan from manager and sends back confirmation.
2. From task list [**6 7**] sent by manager, programmer #1 chooses task **6**.

   ACTIVITIES: PX-1.a, PX-1.b

   QUESTIONS: F4b
3. From task list [**6 7**] sent by manager, programmer #2 chooses task **7**.
4. Programmer #2 attempts a direct change to UI_VT. This is an invalid access request and should be flagged by the system. If the attempt is denied, consider programmer #2 sending a change request to Team 3.[4]

   PERFORMANCE MEASUREMENT: Note action of system.

   ACTIVITIES: PX-1.b, PX-2.b

   QUESTIONS: F4d, F4j, F5c, S3
5. Both programmers report progress and resource consumption at end of week 2.
6. Programmer #1 makes change to code, tests, sends new version to Q/A. (Programmer #2 is working in parallel and should not see the new version.)

   ACTIVITIES: PX-1.c

   QUESTIONS: F5j
7. Programmer #2 makes change to code, tests, sends new version to Q/A. Notifies manager.

   ACTIVITIES: PX-1.c

7.3.7. Team 3

This individual is responsible for packages in UI_VT.

1. Receives global plan from manager and sends back confirmation.
2. Accepts task **2** from manager.

   QUESTIONS: F4b

[4]For this experiment, however, programmer #2 proceeds without changes to UI_VT.

3. Makes change to code and tests.
4. Reports progress and resource consumption at end of week 2.
5. Passes new release of UI_VT to Q/A. Notifies manager.
   ACTIVITIES: PX-1.c

7.3.8. Documentation Group

1. Receives plan from manager and sends back confirmation.
   ACTIVITIES: PX-1.a, PX-1.c
   QUESTIONS: F4f
2. Takes one-week vacation, as planned. Upon his return, he wants latest status of his involvement in project.
   ACTIVITIES: PPM-3.e, PX-1.a
3. Accepts design document changes from Team 1.
4. Updates user manual and releases it to Q/A.
   ACTIVITIES: PX-1.c

7.3.9. Q/A Group

1. Receives task from manager. Defines a release note format (or calls up a template from a library) as the procedure for accepting a maintenance release. Adjusts Q/A plans for new release.
   ACTIVITIES: PDM-3.a, PDM-4
   QUESTIONS: F5a, F5b, F5c, F5d, F5f, F5g, F5h
2. Refines plan and sends it back to manager.
   QUESTIONS: F5i
3. Receives two (independent) fixes from Team 2. Performs acceptance test on UI_SM. Integrates this change into the subsystem.
   ACTIVITIES: PDM-3.b
   QUESTIONS: F5f
4. Receives one fix from Team 3. Performs acceptance test on UI_VT. Integrates these changes into the subsystem.
   ACTIVITIES: PDM-3.b
   QUESTIONS: F5f
5. Receives two (bundled) fixes from Team 1. Performs acceptance test on UI_CLI and checks new source against coding standards. Integrates these changes into the subsystem.
   PERFORMANCE MEASUREMENT: Record the elapsed time required for the standards checking tool.
   ACTIVITIES: PDM-3.b
   QUESTIONS: F5f, P1c
6. Receives new user manual from Documentation group.
   ACTIVITIES: PDM-3.b
7. Consider how the Q/A group would report back to the developers if they discovered a problem in the newly-generated document.
   ACTIVITIES: PDM-2.c

QUESTIONS: F5i

8. Performs regression test on subsystem.

   ACTIVITIES: PDM-3.c

   QUESTIONS: F5g

9. Creates a customer deliverable, release 1.1, which consists of the latest executable code and user manual plus a release note (see Figure 7-12), and informs manager.

   ACTIVITIES: PX-1.a, PDM-4

   QUESTIONS: F3e, F4a, F4d, F4g, S2

8. Hints to the Experimenter

The PM experiment has been reviewed and applied to several environments. From that experience, as well as the instructions and procedures used in the first set of evaluations based on the SEI method (documented in [4]), some guidance is in order for the persons responsible for executing the experiments on a particular environment. The guidance to the experimenter fall into two areas: how to go about performing the experiment and how to deal with incomplete coverage of functionality.

8.1. Process of Experiment Development

The application of the PM experiment is not a simple task that can be done by naively following the steps one by one as described in the generic experiment. The experimenter first has to get an overview of the experiment as well as the concepts supported by the environment. This is followed by a high-level design of the experiment steps in terms of the environment. This requires a detailed knowledge of the environment concepts and their implementation. Finally, the command-level and keystroke-level design of the experiment steps can be done. Because the experiment is comprehensive, some steps may seem repetitive, e.g., each team creating new versions. Results of such activities feed into later steps in the experiment and therefore should be performed. However, some of the details in a step may not have to be documented at the keystroke level. On several occasions, the experimenter is given a chance to explore additional features of the environment (this is indicated in the experiment description with the words "Consider ..."). These instances are optional.

The execution of the experiment has some pitfalls as well. It is important to investigate a way of making snapshots of the environment database and allowing restoration of these snapshots. This is especially useful if the experimenter is not very familiar with the environment and is learning about it at the same time he is performing the experiment. Because the experiment was designed to be comprehensive and realistic, many of the functional areas covered by the experiment are interrelated. Therefore, it is difficult to try to first apply the experiment to a subset of the available functionality and then extend it. Another reason is that integrated environments sometimes assume use of full functionality and provide limited support for augmenting an existing database. However, this is the process that an organization will have to go through when new technology is inserted.

A particularly thorny problem in the experiment's execution is the time scale of the experiment. The scenario calls for a series of activities to be carried out by different groups, and for each activity to take some specified amount of time (ranging from days to weeks). However, executing an experiment step requires hours, not weeks; in other words, the "project" does not take place in real time. We do not see any clean way to avoid this problem. On some systems, the experimenter can proceed with the experiment more or less as defined with only the schedule awry. On other environments, it may be possible to change the system clock to reflect changes in simulated time, and consequently get project information that is correctly recorded. As a last resort, the experimenter could telescope the tasks in the experiment to make their durations much shorter. No doubt other tricks could be played.

8.2. Incomplete Functionality Coverage

Most environments will be deficient in some of the facilities exercised by this experiment. If many of the tools commonly used in project planning and control are missing from the environment under study, it is probably not a good candidate for an evaluation of project management features. If, however, the environment has poor support in only a few areas (e.g., there is no cost estimation tool, or no planning tool, or access control is not project-oriented), you can proceed with the experiment and make allowances for those missing or rudimentary capabilities. This section is an attempt to give you some guidance on how to deal with the most likely deficiencies. It is organized by *activity area*.

8.2.1. Project Plan Management

For creating and revising the project plan, make use of whatever planning tools are provided: WBS, scheduling, cost estimation, resource assignments, etc. Where none are provided, assume the planning has been done with some planning tool. Based on the information provided in the scenario, enter and update the planning data manually and perform the plan instantiation activities.

The parallel development of plans and merging of individual plans into a global plan is not supported by most current project management environments. You will have to manually merge individual plans from the members of the teams.

In the PM experiment, what-if analysis is triggered by a change in personnel which forces the manager to juggle the assignment of human resources in order to keep the schedule from slipping. You should run as many other exploratory scenarios as the system and time permits, since the capability to answer what-if questions is extremely valuable. These activities are described in PPM-3, Project Plan Revision.

8.2.2. Plan Instantiation

If an explicit reporting facility is not provided, you will have to write queries to extract resource and status information from the project database.

Planning may not be integrated in the environment, i.e., it supports planning but does not support tasks (activities) in the planning stage. To get around the lack of formal task management, you could have the manager communicate with other project members via electronic mail, sending plans, error reports, etc., and receiving progress reports.

8.2.3. Project Execution

If the system is weak in tying project-related communication to tasks (for example, notifying a project member that a task is completed), this will be a mostly manual procedure of sending electronic mail messages.

An operating system’s access control may not match the "team" model; it is often undesirable to couple access control to project accounting. Most environments have primitives for access control, but some have better support for logical groups and accounts. Some have the notion of a "task description" which automatically links a task with the users assigned to it. If the project management environment has no notion of access control, try to map onto the operating system's access control model and determine the appropriateness of this model.

8.2.4. Product Management

Traceability across life cycle products may be accomplished by the system in a simple-minded way (for example, by naming the related document in the text), or it may not done at all. Similarly, the ability to track project documents as they evolve may be missing or may call for considerable knowledge and effort on the part of the user.

If the environment does not have a mechanism for controlling change requests and bug reports (tracking them, for example, or relating them to designs or code to be changed), this is not a serious impediment to carrying out the PM experiment. Assume that the maintenance group can get by with electronic mail and text files for informing their own members and their customers.

9. Summary

This milestone report is the sixth in a series of experiments designed to apply a rigorous methodology in evaluating Ada environments. In it, we review the methodology, discuss the scope of software project management, define a set of activities and evaluative questions by which to assess a project support environment, and set up a generic experiment organized by the roles commonly found in project management. The design of the PM experiment has been validated by internal review and by partial instantiation on two environments.

10. References

[1] Feiler, P.H., and Smeaton, R.
Managing Development of Very Large Systems: Implications of Integrated Environments.
In *Proceedings of International Workshop on Software Version and Configuration Control*. German Chapter ACM, GI, Siemens AG, Teubner Verlag, Grassau, W-Germany, Jan, 1988.
Forthcoming as an SEI technical report.

[2] Weiderman, N.H., Habermann, A.N., Borger, M., and Klein, M.
A Methodology and Criteria for Evaluating Ada Programming Support Environments.
SEI Annual Technical Review :17-26, 1985.

[3] Weiderman, N.H., Habermann, A.N., Borger, M., and Klein, M.
A Methodology for Evaluating Environments.
In *2nd ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments*, pages 199-207. ACM, December, 1986.

[4] Weiderman, N.H., et al.
Evaluation of Ada Environments.
Technical Report CMU/SEI-87-TR-1, Software Engineering Institute, March, 1987.

[5] Borger, M.W., Feiler P.H., Klein, M.H., and Weiderman, N.H.
Evaluating Ada Programming Environments: Lessons Learned.
SEI Annual Technical Review :13-26, 1987.

[6] Borger M.W., and Weiderman, N.H.
Generic Evaluation Experiments for Assessing Ada Environments. Support of Configuration Management Activities.
In *Proceedings of the Ada Europe Conference, Stockholm, Sweden*. May, 1987.

Acknowledgements

This report has been reviewed internally by other members of the Evaluation of Ada Environments project, including Bob Ellison, Susan Dart, Cliff Huff, Dan Miller, and Marc Graham, as well as other members of the SEI technical staff. It has also been sent outside the SEI to Cordell Greene of the Kestrel Institute, Vic Stenning of Imperial Software Technology Ltd., and Yosi Amram of Rational. We especially wish to thank Bob Glushko of the SEI and Ram Banin of Atherton Technology for their editorial comments. Our report has benefited from the expertise of these individuals. Finally, we wish to acknowledge Dan Miller's work in instantiating the PM experiment on the ISTAR environment.

Appendices

Appendix A, **Functionality Checklist**, is a cross-reference between activity (column one) and experiment step (column 2). A Y in column 3 means "yes, the activity is supported by the environment." Column 4 is reserved for comments or further explanation about a facility in the environment. The following abbreviations apply:

ACRONYM	MEANING/ROLE	SECTION
Setup	Set-up for experiment	7.3.1
CU	Customers	7.3.2
MPM	Manager for Product Maintenance	7.3.3
SA	System Analyst	7.3.4
T1	Development Team 1	7.3.5
T2	Development Team 2	7.3.6
T3	Development Team 3	7.3.7
DOC	Documentation Group	7.3.8
QA	Quality Assurance Group	7.3.9

Appendix B, **Cross-Reference Table for Execution of Experiment** associates questions with experiment steps within criteria area. The same abbreviations are valid. The purpose of this table is to help complete the answers to questions by showing which experiment steps contribute to the answer.

Appendix C, **Cross-Reference Summary Matrix**, shows at a glance the phases and steps in the evaluation methodology as it relates to the project management experiment.

Appendix D, **Performance Questions Within Experiment Steps** shows which experiment steps ask questions about performance and in which area (elapsed time, memory space, responsiveness).

Appendix E, **Activities and Plans (Global View)** is a chart of the entire project management scenario.

Appendix F, **An Illustration of Documenting an Instance of the PM Experiment** illustrates one project management tool: MacProject for the Macintosh.

Appendix A: Functionality Checklist

	Step #	Supported (Y,N)	Observations
Project Plan Management			
Project plan creation			
tailor planning support facilities	Setup-6,Setup-7,Setup-8		
link plans to baseline	MPM-5		
develop WBS	MPM-5		
estimate work cost	MPM-5		
develop schedule	MPM-5		
assign resources	MPM-9,T1-1		
estimate project cost	MPM-5		
merge group plans into global plans	MPM-7		
generate plan document	MPM-5,MPM-8		
Project monitoring			
report on actual progress	MPM-10,MPM-13		
analyze progress against schedule	MPM-10,MPM-13		
compare actuals to estimates	MPM-10,MPM-13		
analyze resource utilization	MPM-10,MPM-13		
generate summary reports	MPM-10,MPM-13		
Project plan revision			
baseline the plans	MPM-11		
perform what-if analysis	MPM-11		
handle schedule slippage	MPM-11		
handle personnel changes	MPM-11,DOC-2		
handle changes in WBS	MPM-12		
handle changes to project structure	MPM-12		
handle changes in deliverables	MPM-12		
adjust costs based on actuals	MPM-12		
handle computing resource changes	MPM-12		
generate reports	MPM-11		
Plan Instantiation			
Project installation			
set up product structure	Setup-1,Setup-2,Setup-3,Setup-4,Setup-5		
set up team structure	MPM-9		
set up task structure	MPM-6,MPM-9		
Reporting mechanism installation			
set up task completion reports	Setup-8		
set up accounting reports	Setup-8		
set up statistical reports	Setup-8		
Reflecting modifications to plan			
reassignment of people	MPM-11,T1-5		
changes in task structure/schedule	MPM-12		
changes in project or product structure	MPM-12		
changes in computing resources	MPM-12		

Project Execution

Communication and coordination	
communication between team members	MPM-11,MPM-16,T1-1,T1-2,T1-5,T1-9, T1-13,T2-2,QA-9,DOC-1,DOC-2
work area coordination	T2-2,T2-4
task completion and notification	SA-9,T1-4,T1-13,T2-6,T2-7, T3-5,DOC-1,DOC-4
Information access and control	
project database access	T1-5
access control	T1-5,T2-4

Product Management

Traceability of project documentation	
access trace information	SA-4
creation of trace information	Setup-3
Control of change requests	
approve requests	MPM-4,SA-1
log and track requests	CU-5,MPM-1,MPM-15,SA-1,SA-9,QA-7
display change history	MPM-15
Quality control	
check adherence to standards	QA-1
V&V and acceptance testing	QA-3,QA-4,QA-5,QA-6
support test plan development	QA-8
Product Release Control	MPM-15,MPM-17,QA-1,QA-9

Appendix B: Cross-Reference Table for Execution of Experiment

Appendix C: Cross-Reference Summary Matrix

Appendix D: Performance Questions Within Experiment Steps

Appendix E: Activities and Plans (Global View)

Appendix F: An Illustration of Documenting an Instance of the PM Experiment

A description of the architectural model is expected in Phase 4 of the methodology. To give guidance to the experimenter in this phase, we present as an example the following description of a typical project management tool, MacProject™. The description includes an overview of the functionality and user interface of MacProject, a completed Functionality Checklist, and an example of the sequence of operations for one step of the experiment.

1. Overview

MacProject is a project management tool for the Apple Macintosh computer.[5] With it, you can visually schedule and track a project, its resources, and associated costs; update a project plan with changes as they occur; and perform what-if analyses.

Functionality

To set up and monitor a project, MacProject offers a variety of representations or views:

- Schedule Chart—Lays out a project, its tasks and milestones and their interdependencies, plus resource and time information about each task in a PERT chart (network diagram) format. All projects begin as Schedule Charts, and most project information is entered or modified here.
- Resource Timeline—For each resource, shows all tasks ordered by their start dates, in the form of a GANTT style bar chart. You can add task information or set dates in this view.
- Task Timeline—Shows all tasks in the order in which they occur, as a GANTT style bar chart. You can add task information or set dates in this view.
- Task Cost Entry—Enters fixed costs and income.
- Resource Cost Entry—Enters unit costs for each resource.
- Cash Flow Table—Shows cash on hand; both incremental and cumulative costs and income are shown.
- Project Table—Shows all information about each task in a project: start and finish dates, duration, cost, resources required.

Tasks are represented in the Schedule Chart by boxes. Each task has its own task information: a name, a duration, one or more resources (equipment, personnel), and a cost. Potential over-assignment of resources is not disallowed or checked. *Costs* may be fixed costs or ongoing costs such as salaries. Seven types of information (dates, cost, etc.) and up to four resources are available for displaying at the corners of each task box, but no more than four items of information can be displayed at once. Besides tasks, MacProject lets you show *milestones* (key events or progress markers), dependencies, and *dates*. You can set a date for a task or milestone, or you can let the software calculate dates; the choices are earliest start, earliest finish, latest start, latest finish. MacProject will calculate and display critical paths as well as slack time for tasks not on the critical path.

The products needed by an activity and the products produced by an activity are not represented.

[5]Currently, MacProject is not portable to other computers.

The arcs or links between tasks can only be annotated by manually attaching text, i.e., the software does not associate text strings with links. So, for example, it would be awkward to model contract deliverables.

The project duration scale is under user control. You can set the display of durations and dates to be minute, hour, day, week, or month. The time interval for the Timeline charts and the Cash Flow Table can likewise be set to any of the above, as well as to biweekly or quarterly.

MacProject has a calendar that can be edited to reflect working hours, holidays, etc. It is integrated into the project management software so that scheduling, for example, will take into account those days that employees are unavailable. Validity checking on dates is automatic (for example, if you set a date to a non-working day, the system will prompt with a warning message). There is only one calendar, and calendar settings affect the entire project.

The temporal order of tasks is expressed through layout constraints (the left-to-right flow of tasks). In this way MacProject prohibits *loops*, tasks connected in a circular fashion.

User Interface and Documentation

The Macintosh has a deservedly fine reputation for its graphical user interface. Such features as icons and menus (which provide visual feedback), manipulation of images via the mouse, and abundant on-line help and guidance make MacProject easy to learn and use. In MacProject, the seven views of a project are linked and are kept in sync with each other. Switching between views is as easy as selecting an alternate format from a menu. The usual Macintosh editing commands are available in MacProject: select, cut and paste, a variety of fonts and styles for text, commands for printing and viewing on the screen, etc. In the Macintosh style, many common commands have abbreviations (for example, control-T to show task information).

A section in the manual called "Steps in creating a project" describes, with simple text and diagrams, how to set up a project on the computer.

Chapter 3 of the MacProject manual is called "Managing Projects." It offers hints and explanations by answering common questions about project management such as:

- How can I finish the project sooner?
- Which tasks should I shorten?
- What if I lose or add staff or equipment?
- What if I change working hours or days?
- What if there’s a deadline in the middle of the project?

For example, "How can I finish the project sooner?" suggests the following methods for trimming a project’s schedule: eliminate a task on the critical path, replan dependent tasks to be done in parallel, overlap sequential jobs, decrease the duration of tasks on the critical path, or increase the number of work days or work hours.

Limitations of MacProject

While MacProject is a very useful tool for the project manager, it has limitations. For very large and complex projects (hundreds of tasks), MacProject would probably not be suitable because of speed and memory constraints of the Macintosh, and because of limits within MacProject on the maximum number of tasks and resources. Also, MacProject is for project management in general and is not specifically designed for controlling *software* projects.

Other limitations (capabilities that would be desirable and are lacking) include:

- No automatic update to the schedule. Information regarding progress on the project that is passed to the plan management facility necessitates manual updates.

- No support for multiple levels of detail in the Schedule Chart.
- No support for synchronizing plan development by multiple people or for merging plans.
- No shared database.
- No formal support for distributed project planning.

Broadly speaking, MacProject is good for planning-in-the-small but not good for planning-in-the-large, the "grand view" of a project at a global level.

2. Functionality Checklist

	Step #	Supported	Observations
Project Plan Management			
Project plan creation			
tailor planning support facilities	Setup-6,Setup-7,Setup-8	N	
link plans to baseline	MPM-5	Y	
develop WBS	MPM-5	Y	
estimate work cost	MPM-5	Y	
develop schedule	MPM-5	Y	
assign resources	MPM-9,T1-1	Y	
estimate project cost	MPM-5	Y	cost estimation is unsophisticated
merge group plans into global plans	MPM-7	N	
generate plan document	MPM-5,MPM-8	Y	
Project monitoring			
report on actual progress	MPM-10,MPM-13	Y	
analyze progress against schedule	MPM-10,MPM-13	Y	mostly manual
compare actuals to estimates	MPM-10,MPM-13	Y	mostly manual
analyze resource utilization	MPM-10,MPM-13	Y	limited
generate summary reports	MPM-10,MPM-13	Y	
Project plan revision			
baseline the plans	MPM-11	N	
perform what-if analysis	MPM-11	Y	
handle schedule slippage	MPM-11	Y	
handle adjustment in working hours	MPM-11	Y	
handle personnel changes	MPM-11	Y	
handle changes in WBS	MPM-12	Y	
handle changes to project structure	MPM-12	Y	
handle changes in deliverables	MPM-12	N	
adjust costs based on actuals	MPM-12	Y	
handle computing resource changes	MPM-12	N	
generate reports	MPM-11	Y	
Plan Instantiation			
Project installation		N	
Reporting mechanism installation		N	
Reflecting modifications to plan		N	
Project Execution			
Communication and coordination		N	
Information access and control		N	
Product Management			
Traceability of project documentation		N	
Control of change requests		N	
Quality control		N	
Product Release Control		N	

3. Sequence of Operations

The following is an example of the responses the experimenter might give in step MPM-5 of the experiment when evaluating MacProject.

5. Turn remaining recommendations into an initial global plan.

PERFORMANCE MEASUREMENT: *For each of the following sub-steps, record the responsiveness of the tool or facility used in plan development. Record the storage cost of each object in the global plan: WBS, schedule, PERT chart, cost estimate, etc. Note how long a critical path analysis takes as an indication of interactiveness.*

In MacProject, the various views of a project are kept in a single document. The size of this document is 3K bytes. Entering data into any view or updating data is rapid. Changing views, by selecting a new view with the mouse, takes only a few seconds.

a. define work packages in the initial WBS for three maintenance teams, a documentation group, and a Q/A group.

When you begin a new MacProject document, the default view is Schedule Chart. Enter each task by drawing a box on the screen with the mouse and typing a name for the task inside the box. Connect the task boxes. Change the starting, ending, and reporting tasks to milestones.

b. estimate man-days, number of resources (persons), number of days, number of changed lines of code.

Still in the Schedule Chart. For each task in turn, select "Show Task Information" and enter the number of days and then the persons (role) assigned to that task. Select "Show Dates" and set to ON the display of duration, a date, and a resource.

c. perform a cost estimation and set up a budget for the project as well as for teams.

Enter salaries for the technical staff into the Resource Cost table, and fixed costs into Task Cost table. Expenditures will show up as costs in the Cash Flow table and the right-hand column will give cumulative costs.

There is no macro-level tool for cost estimation, and support for budgeting and forecasting is minimal.

d. work out an initial global schedule.

The schedule, in the form of a PERT chart, is maintained by MacProject as tasks are entered and linked into the network. MacProject calculated the critical path in approximately one second.

e. generate a document containing the initial global plan. If possible, generate different views of the plan information, e.g., PERT chart, work package listing, resource chart.

For each of the seven representations available, (1) change the display to that view and (2) select "Print" which produces a hardcopy listing of that view of the plan.

f. retain a version of the plan as part of the project history.

Get an empty folder, name it, and drag the MacProject document into it.

Archiving and CM are not available in MacProject.