Technical Report

CMU/SEI-88-TR-11
ESD-TR-88-012

Managing Development of Very Large Systems: Implications for Integrated Environment Architecture

Peter H. Feiler
Roger Smeaton
May 1988

Technical Report
CMU/SEI-88-TR-11
ESD-TR-88-012
May 1988

Managing Development of Very Large Systems Implications for Integrated Environment Architecture

Peter H. Feiler
Evaluation of Environments Project

Roger Smeaton
Resident Affiliate,
Naval Ocean Systems Center

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

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

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

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Karl H. Shingler SIGNATURE ON FILE
SEI Joint Program Office

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

Managing Development of Very Large Systems: Implications for Integrated Environment Architectures

Abstract: Version and configuration control are mechanisms for managing source code and system builds. In the development of very large systems, built by large teams, development management is the dominant factor. In this paper we examine management support for development through integrated environments and investigate the implications for environment architectures. We do so by defining a project scenario that is to be performed with integrated project support environments. The scenario has been carefully designed to not only determine the scope of management functionality provided by a particular environment, but also to probe implications for the architecture of environments. The implications discussed in this paper are: focus on user activities; the integration of project management and development support concepts; the ability to reinforce and avoid conflict with particular organizational models; the ability to support evolution and change of the product, environment, and organization; and the capability for adaptation and insertion into a work environment. The scenario is part of a methodology for evaluation of environments currently used at the Software Engineering Institute.

1. Introduction

The management and control of the development of very large software systems is a key to their success. As software systems and the projects developing and maintaining them become larger, project management plays a more dominant role. Management comprises management of the product (as accomplished through version and configuration control as well as system modeling), management of the process (e.g., task management and change request procedures), management of resources (project planning and control), and management of the environment (such as introduction of new versions of tools and migration to new hardware).

Version and configuration control, together with system modeling, concentrates its efforts on managing source code and automating system builds. Mechanisms are provided for keeping a history of source code modules, for creating stable configurations of the software in the form of baselines and releases, and for controlling and coordinating the update of source code by multiple developers. These mechanisms work well for small teams of developers. For very large systems, however, the development takes on characteristics to which version and configuration control mechanisms and tools do not easily adapt. Very large systems tend to have multiple major threads of development; releases of software components (check-ins) tend to require several levels of visibility; large development teams require different organizational structures and more personnel-related activities such as training, replacement, and communication; task organization, review, and tracking dominate the project activities. These issues have traditionally been addressed through project management techniques. It is for this reason that we examine environment support for very large system development from a management perspective with emphasis on project management.

In many environments in the past, tools supporting managers in project planning and control were relatively isolated from tools for developers. Tasks were delegated and progress reported back in a manual fashion, and developers lacked facilities for managing their tasks. In the last few years, the concept of an integrated project support environment (IPSE) has received attention. An IPSE provides support for both development and management of projects integrated into one environment; it also allows integration of development tools from external sources. Prototypes of such environments are becoming available [6].

In this paper, we examine management support for development through integrated environments and investigate the implications for environment architectures. We do so by defining a project scenario that is executed on integrated project support environments. This scenario is part of an experiment in the methodology for evaluation of environments currently under way at the Software Engineering Institute [16, 17, 18]. The methodology has been applied to a variety of environments, and the scenario has been carefully designed to not only determine the scope of management functionality provided by a particular environment, but also probe implications for the architecture of environments. Implications in five areas are discussed in this paper: focus on user activities; the integration of project management and development support concepts; the ability to reinforce and avoid conflict with particular organizational models; the ability to support evolution and change of the product, environment, and organization; and the capability for adaptation and insertion into a work environment. These implications can be viewed as requirements for integrated project support environments.

The paper proceeds as follows. In Section 2, we discuss the scope of project and development management by defining the covered management areas and by describing both the context for an experiment scenario and the scenario itself. Section 3 contains a discussion of the implications of management support for very large projects on an environment architecture if the environment were to support the activities in the scenario in an integrated manner. The paper concludes with some remarks about the ability of today's IPSEs to provide integrated project support, along with a summary of additional requirements on integrated environments.

2. Project and Development Management Support

This section describes the scope of project and development management support and the scenario that we use for investigating the requirements. First, we describe the four areas of management support. Then, we give the context for the scenario and conclude by discussing the scenario.

2.1. Scope of Project and Development Management

We have identified four areas of management support:

- management of the product
- management of the process
- management of the resources
- management of the environment

Management of the product is comprised of version control, configuration control, product release control, and traceability of product components. Version control refers to the maintenance of successive, parallel, and derived versions of components, explicit and automatic version selection and retrieval, and coordination of modifications to versions of a component. Configuration control refers to system modeling, i.e., description of the composition of a system; system construction, i.e., build and rebuild of systems as well as the building of hybrid systems with a mix of versions; and derivation management, i.e., cost-effective maintenance of objects derived from other objects through application of tools. Product release control represents facilities for baselining and packaging product deliverables (e.g., executables together with online help and user manuals), for maintaining product release and customer information, and for rolling back to older releases. Traceability of product components refers to the ability to show the relationships between documents describing product components, to trace their relationship through the development, and to browse and query the relationships. Examples of such relationships are those between requirements and specification for components and their code, and between user documentation and the system it documents.

Management of the process represents standards, procedures, and protocols to accomplish orderly and consistent development of a software system. It includes change management, quality control, and task management. Change management deals with changes to a released system version and relates error reports, change requests, and assignments to versions of product components. Quality control specifies various criteria—in the form of standards, validation and verification procedures, and acceptance tests—that must be satisfied at certain times of the development process. Task management refers to facilities for managing task assignments, tracking user activities as part of the tasks, recording and reporting completion of activities, and activating dependent tasks.

Management of resources refers to the activities of project planning and control. On the project manager's side, these activities include: planning activities such as development of work breakdown structures, cost estimation, resource assignment, and scheduling; monitoring activities such as analysis of schedule and resource consumption of actuals against estimates; and plan revision activities to reflect changes such as personnel changes, organizational changes, and changes to the product structure. On the development support side, management of resources refers to facilities for instantiating project plans, that is, the ability to reflect product, team, and task structures, for automatic reporting of progress, and for reflecting revisions in the plans.

Management of the environment is concerned with changes to the project support environment as well as the use of and changes to the computing environment. During the lifetime of a project, especially large projects, it is likely that the environment will be changed. New versions of tools will be installed; new tools will be introduced; and tools knowledgeable of organizational procedures will be adapted to reflect adjustments of the procedures. Just as the environment controls the release of new product versions, it is responsible for introducing new versions of tools while maintaining the consistency of the development database. The use of networked computer workstations requires the environment to support distributed operation, but it also requires support for maintaining the environment. This includes distribution, installation, and tracking of the environment configurations on a large number of workstations, and use of server machines to offload processing from each individual workstation.

2.2. Context for the Scenario

The following scenario simulates a project for maintaining a released software system. The product consists of several subsystems, one of which is the user interface (UI) subsystem illustrated in Figure 2-1. The organization responsible for maintaining and enhancing the product is split into two groups: a maintenance group and an enhancement group. The maintenance group fixes problems with the current release, while the enhancement group extends the functionality of the product. The organization, which is fictitious but representative, is illustrated in Figure 2-2. For the purpose of this paper, the scenario concentrates on the activities of the maintenance part of the organization.

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 in cooperation with team leaders. The plans are approved, and the project is carried out by several teams, a documentation group, and a quality assurance (QA) group. During the life of the project, its progress is monitored by the project manager and the team leaders. 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.

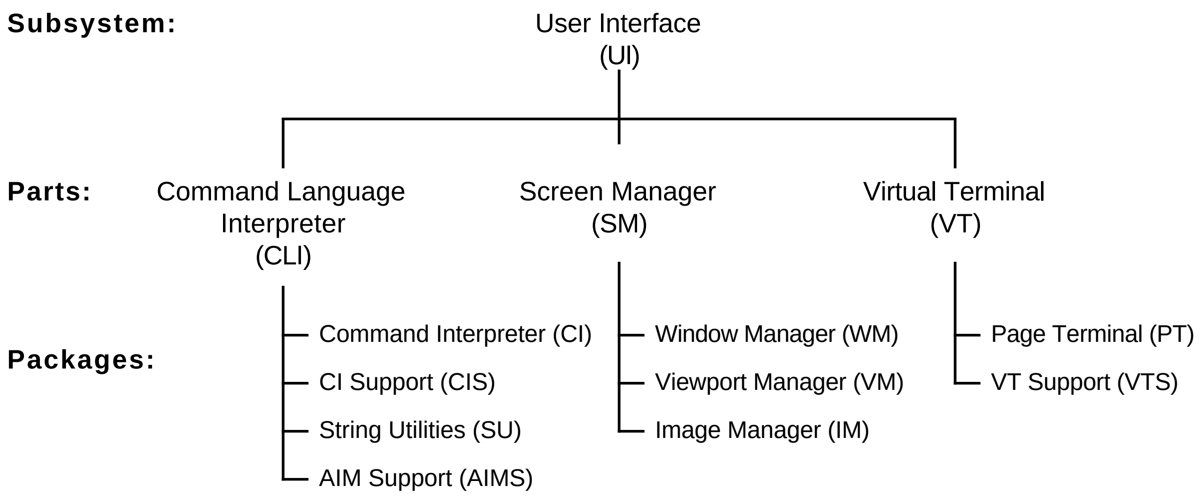

Figure 2-1: Software System Structure

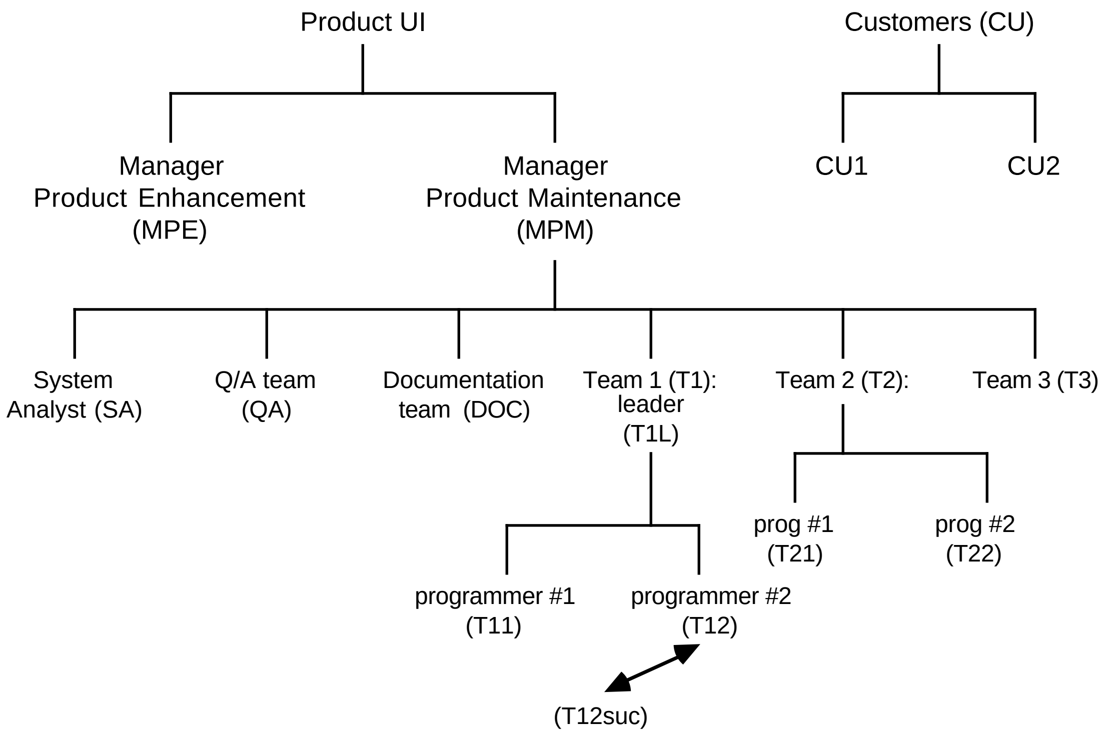

Figure 2-2: Organizational Structure

We are investigating the ability of the environment to support different organizational structures and management styles. Therefore, the maintenance group is organized as follows. It consists of a manager, a system analyst, and five teams: three maintenance teams (one for each part of the software system), a documentation group, and an integration and quality assurance group. The manager is responsible for the overall planning and management of the maintenance activities. Although some of the manager's activities may be those of an administrative assistant, we refrained from introducing another role in order to keep the experiment at a manageable size. 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 pair of 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—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 QA group has the responsibility of working out a QA plan, performing acceptance tests, integrating team deliverables, and packaging the new release.

2.3. The Experiment Scenario

The scenario of the development management experiment is illustrated in Figure 2-3. The label in the upper left of a task box indicates who is responsible for the task (using the abbreviations from Figure 2-2). The task is described in the task box. The arrows connecting task boxes are marked with deliverables. The design and code change task for Team 1 is expanded into more detail, illustrated through a bubble in Figure 2-3. An expansion of detail for the tasks of Teams 2 and 3 and the QA team is not essential for the purpose of this paper.

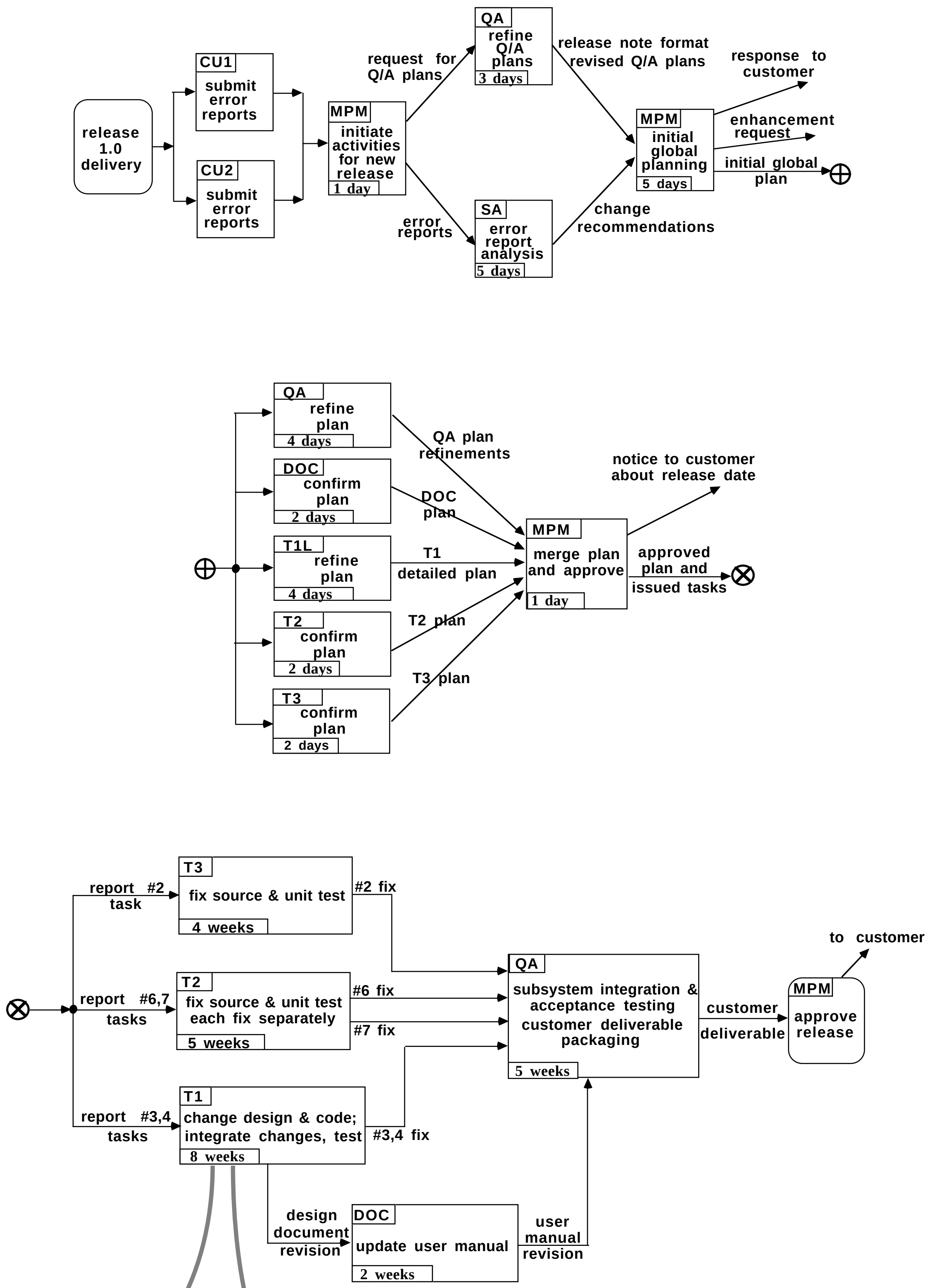

Figure 2-3: Scenario

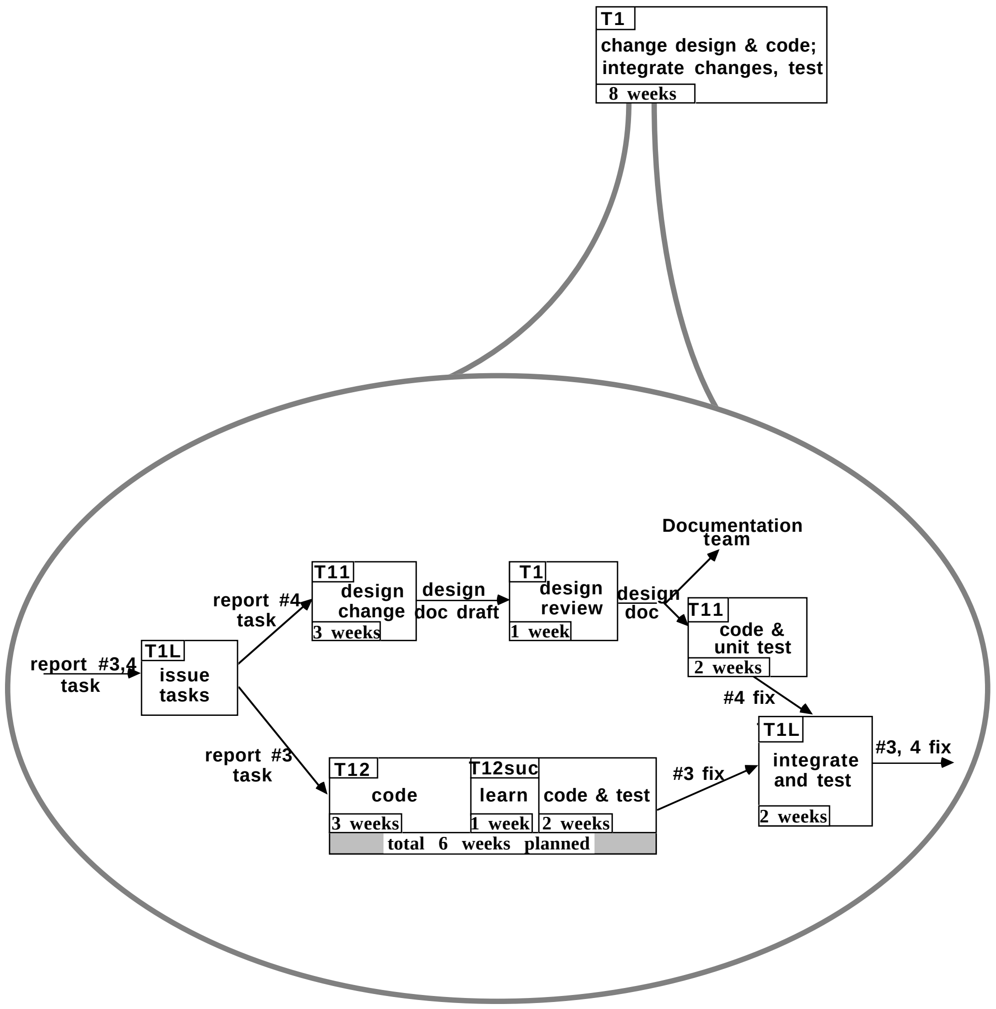

Figure 2-4: Scenario Detail

Two customers have filed a number of error reports. We have chosen the collection of error reports and their implications in a way that permits 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 and releases containing the fixes, relating error reports to each other, and relating responses to error reports.

The manager initiates maintenance activities which will culminate in a new release. He requests that the quality assurance group refine a quality assurance plan for the next release, and he sends the error reports to the systems analyst. The system analyst classifies the error reports, examines the development history and the design, specification, and requirements documents to locate the cause of each error. The analyst then determines the scope of corrective changes by querying dependency information and specifies the necessary actions. QA develops a quality assurance plan along with standard forms and checklists. In this part of the scenario, the manager should be able to issue these planning tasks with little overhead.

The manager makes an initial global plan, which is passed to team leaders for refinement. This plan includes task descriptions to implement bug fixes, based on the change recommendations from the systems analyst. The team leaders report back their refinements, which the manager merges into one plan. The plans are then approved by the manager and instantiated. This planning activity examines the environment's ability to support coordination of multiple persons working on a plan. 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 permit the members of Team 2 to choose among assignments.

Team 1 has changes that require an update to documentation, so 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. One programmer (T12) leaves in the middle of his task and is replaced by a successor. Because the new programmer will take a week to become familiar with the task and the task is on the critical path, the schedule will slip. An investigation of plan alternatives results in the conclusion that no slippage is necessary if the new programmer does not participate in the design review. This tests the visibility of the slippages in the critical path and investigates how well the project planning copes with plan changes and conflicts.

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

QA accepts deliverables from three teams and merges the new versions into an integrated subsystem. Figure 2-5 illustrates the resulting version history of subsystem UI. The version graph for UI shows the relationship between the different versions and the teams involved in creating them. Figure 2-6 shows the version thread through component versions for each version of UI. The components shown are themselves composed of smaller components (as shown in Figure 2-1) with their own version history. This exercises evolution of versions of a system that has a multi-level configuration structure.

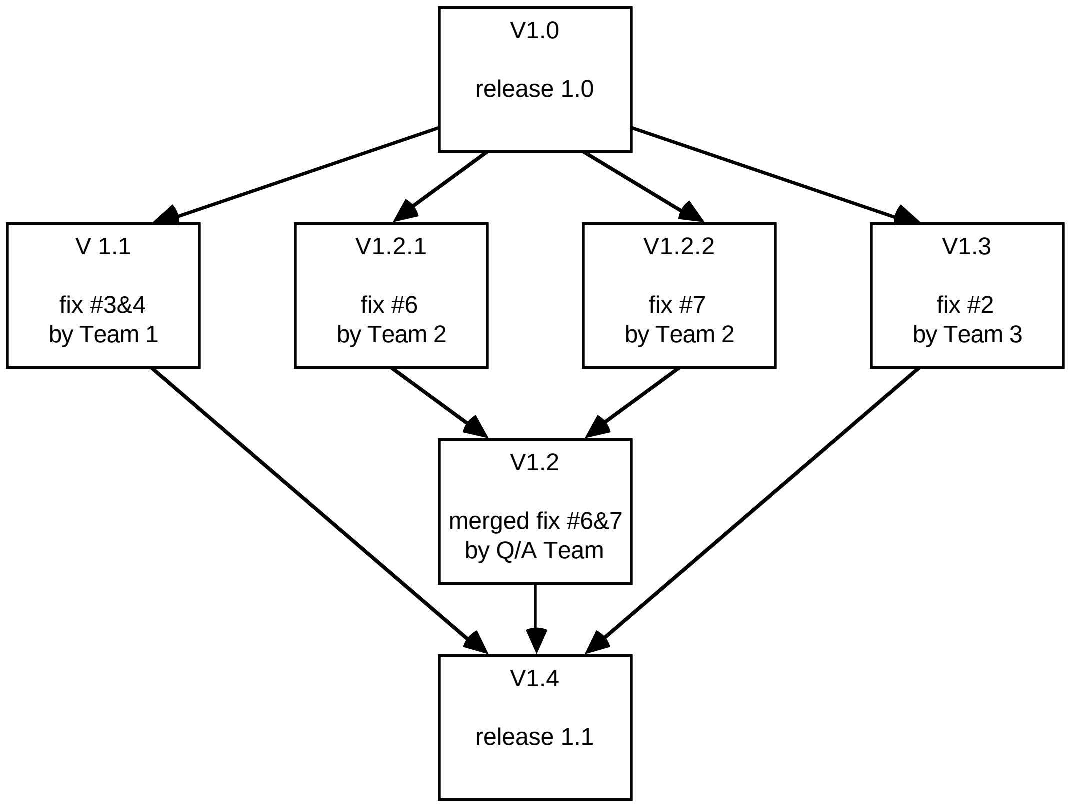

Figure 2-5: Version History of UI Subsystem

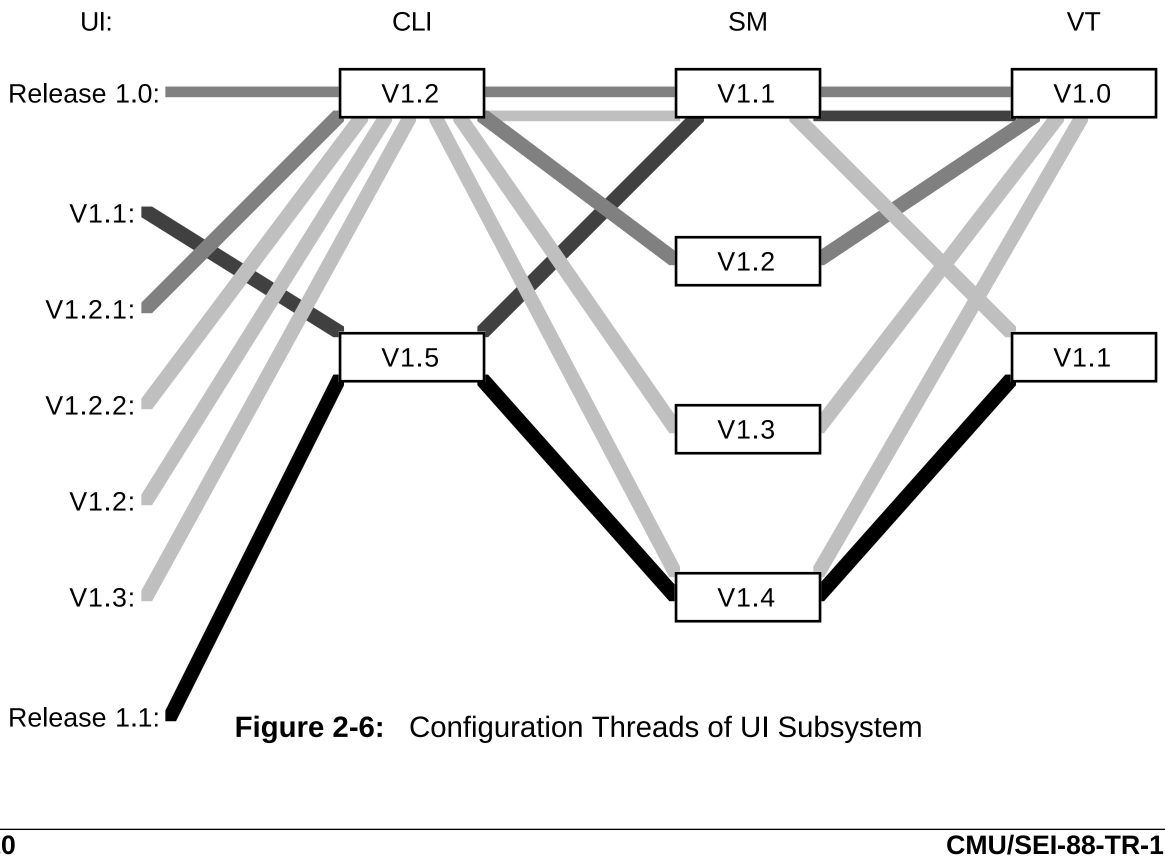

Figure 2-6: Configuration Threads of UI Subsystem

QA receives documentation and completes a release document. The manager signs off on the new release, and it is distributed to appropriate customers. The experiment closes with the manager informing all project members of a celebration of project completion, investigating informal communication support in the environment.

3. Implications for Environment Architectures

We have designed this scenario in such a way that it can be used as an experiment to evaluate the functionality provided by an environment. At the same time, the scenario probes issues related to environment architectures when management support is integrated into an environment. In this section, we discuss five issues that we view as crucial ones for IPSEs to address in order to become accepted. These are:

- Raising the level of abstraction of environment operations to more closely reflect actual user activities.
- Integrating project management and development support in a manner that benefits both the manager and the developer and does not unduly increase the complexity of the environment.
- Understanding and supporting various processes, procedures, and protocols without imposing its own; and automating steps where appropriate, thus reducing the responsibilities of the user and the complexity of the environment.
- Supporting evolution and change, not only of the product but also of the development process and its management, along with change to the environment itself.
- Adapting the environment to the needs of the organization to be supported.

3.1. Focus on User Activities

The scenario described in Section 2 is defined in terms of user activities, which have been specified in a manner that is independent of a particular environment or collection of tools representing an environment. By specifying the user activities abstractly in terms of concepts familiar to the user rather than concepts and operations that exist in some environments today, we are able to determine the complexity and amount of interaction required to accomplish a task in a particular environment. To illustrate, consider the difference between the *user activity* of replacing one person on a team with another person and the *operations* that must happen at a lower level to carry out this change—for example, changing the ownership of files and directories, or moving messages containing task descriptions from one mailbox to another through file copy operations. In other words, we are able to determine how closely the environment models and supports concepts and operations of the user's domain. Systems such as the Apple Macintosh have demonstrated that learnability and ease of use improve considerably when concepts from the user domain are directly supported.

3.2. Integration of Project Management and Development Support

Integration of project management and development support into one environment can benefit both components of the environment. On one hand, development support concepts and operations can be made available to project management facilities. On the other hand, the introduction of some project management concepts into the development support facility can also benefit the developers. Finally, the integration can reduce the number of concepts that the user is exposed to, which in turn can reduce the complexity of the environment from the user's perspective. In the following paragraphs we further discuss each of these points.

Development support usually provides facilities for maintaining versions of objects (such as design documents, source code, or user documentation) for grouping selected versions together into configurations and for coordinating access and modification of the objects by multiple users. In an integrated environment, these facilities can be made available to project management. Objects in this domain (such as work packages, schedules, resources, plans, task descriptions, progress reports, change requests, and error reports) should have the same status as objects in the development domain. As many of these objects change over time, it is desirable to keep a history in the form of versions. Users may interact with multiple versions of one object simultaneously, for example, when a manager does what-if analysis with alternative plans. Capabilities for highlighting differences between versions of objects are also desirable, e.g., when alternative plans are compared or when change requests are examined for amendments. Project plans may be decomposed and worked on by several people, requiring a well-defined interface between the plan components, along with coordination and merging of updates to the plan components. ISTAR is an example of an environment in which objects from both the project management and development support domains are managed in the same manner [2], i.e., objects from both domains can be removed and passed around.

Integration of project management and development support requires that management concepts are supported in the development facility. In addition to facilities for managing the product, mechanisms must be available to support various team structures and the management of tasks. From a management perspective, team structures can be recursive, i.e., a team can consist of a set of teams reflecting a management hierarchy. This means that a person can be viewed as a member of a team as well as a member of its enclosing teams. A person can also be a member of more than one team, e.g., a member of the user interface team and a member of the device handler team. Team structure support in the development facility of the environment should provide access control, accounting (of computing resource usage), and communication (e.g., sending an announcement to all members of a particular organizational entity). Access control and accounting mechanisms provided by the underlying operating system may not be appropriate to model the desired team structures. Information about the team structure may have to be maintained by the user in several places, e.g., in access groups, in account structures, in e-mail distribution lists, and in electronic bulletin boards (one per team). Task management can take several forms, some emphasizing support for the developer, others leaning more to supporting managers. Task management allows developers to use simple task descriptions to keep track of tasks to be done. Completed tasks are removed from a user's task list but may be kept as a record. While working on a task, the user may log various actions taken, and the environment may automatically add information to the task log. Apollo DSEE [9] provides such a facility, automatically logging check-in of modified objects. The task management mechanism may automatically propagate task completion, either by notifying a person or other task or by automatically activating tasks that depend on the completion of the task (i.e., on one of its deliverables). Alternatively, new tasks may be activated when an object such as a new version of a source code module is checked in—as in DSEE's monitor capability. Tasks may be organized into hierarchies, i.e., tasks can be partitioned into subtasks. The concept of task may be more formal if it also specifies the objects required to do the task and the deliverables; the task may represent a working environment and protection domain. The ISTAR contract model [3] is an example of such a task management mechanism embodying more management control.

An integrated environment can attempt logical integration, that is, integration at the conceptual level. The result may be a reduction in the number of concepts the user is exposed to and a reduction in the amount of redundant information the user must maintain or access. For example, work packages from the work breakdown structure may become schedulable entities. Once accepted as part of a plan, they may become task descriptions—without redescribing the task. When progress is reported through timesheets, the environment can automatically record all tasks the user worked on, including terminated tasks, rather than requiring the user to report progress in terms of job codes. In contrast to an environment that is a collection of independent tools, an integrated environment reduces cognitive demands on the user.

3.3. Process Modeling and Automation

A development project is managed through well-defined processes, procedures, and protocols. They provide a certain amount of order and consistency to the evolution of the product. Examples of product-oriented processes are acquisition of user requirements, consistent changes to source code, and consistent update of derived objects (e.g., object code). Examples of management-oriented processes are use of certain development methods in a particular style, document review and approval, change request control, and prescribed patterns of information flow and decision making in the organization. Environments of the toolkit variety provide little support for the process; users are expected to use the tools according to certain conventions in order to maintain consistency and order. However, as development support environments and IPSEs attempt to capture various processes and support them, two issues are raised: conflicts between the processes modeled by the environment and the models given in the organization; and appropriate automation of various processes to assist the users. In the remainder of this section we elaborate on each issue.

Many management tools and facilities, whether they are for configuration management or project management, incorporate process models. These models may inadvertently restrict the use of the tool due to conflicts with the process intended to be supported. For example, a task management facility supporting hierarchical tasks may be considered too restrictive because it implies a hierarchical organization. However, the task management mechanism is more flexible if the hierarchy reflects only the creation of tasks and the tasks can communicate (i.e., pass on deliverables) independent of the creation hierarchy. Similarly, the resource assignment facility may be too restrictive if it requires assignment of individuals to tasks. A cooperative team management policy, as found in Team 2 of the scenario, is difficult to support. As these examples demonstrate, it is important for an environment architecture to separate management support mechanisms from policies. Furthermore, policies, protocols, and procedures differ from organization to organization, and the capability to tailor them is desirable.

Formal encoding of processes permits the environment to automate steps of the process. The environment can take responsibility for certain actions. It can be responsible for consistently updating derived objects to reflect their source (e.g., the UNIX *make* facility) and for consistently maintaining redundant information while requiring the user to enter or modify it only once [7]. The environment can inform the user of related documents that are affected by change (e.g., DSEE monitors [9]) and can determine if acceptance criteria are satisfied. It can enforce protocols (e.g., submission and approval of a change request before the actual change can occur); it can control the actions different people can take (e.g., Darwin [11]); it can guide the user through steps of the protocol (e.g., inform the user that a particular test suite must run before a modified module can be released [13]); and it can be responsible for certain actions that are part of the protocol (e.g., automatically notify the issuer of a task and activate dependent tasks upon completion of a task). This automation can reduce the perceived complexity of the environment. However, full automation is not desirable. For example, successful execution of a test suite on a modified program should not automatically declare a task completed and trigger notifications and task activations. Similarly, slippage in a low-level task on the critical path should not necessarily result in automatic notification of all managers in the management hierarchy. Automation must be properly applied, and the user should be able to control the automation where appropriate [8].

3.4. Support for Evolution and Change

Software development and maintenance is a process of evolution and change. An environment can support development adequately only if it anticipates and facilitates change. The discussion in this section first elaborates on changes during the lifetime of a project and then describes how change is supported.

Environments typically support evolution and change by providing version control on source code components, as does RCS [14]. A complete history of changes is recorded, changes are coordinated, and concurrent development is supported on different version branches.

System models and configuration descriptions, (i.e., description of a particular composition of a system) are often maintained as text files (e.g., UNIX *make* [4] and DSEE system modeling [10]). The expressive power of the system modeling notation is concentrated on describing a system rather than describing changes to a system. Support for change is limited to submitting the text file containing the system description to version control. The evolution pattern of system descriptions does not necessarily follow that of source code components; thus, existing version control mechanisms may not appropriately support these change patterns.

The software product consists of more than the source code and the generated executable. The product to be installed on a system may consist of the executable, a help database, a library of descriptions used to tailor the system, and other programs that the system uses. The deliverable may consist of specification and design documents, the implementation, test suites, and user manuals. The environment must provide support to help manage change of the complete product and to coordinate the evolution of the product through several development branches.

Change is not limited to the product. During the lifetime of a project, tools used by the environment and the user may change, i.e., new releases of the tools may be installed. It is important that environments control the installation of new tools. Environments must know when new releases of tools are installed and they must track the effects of the new release on objects generated by earlier releases of the tool. Similarly, process models may change during the life of the environment. New methods may be introduced together with supporting tools, or the procedure for validating and approving a design change may be improved. If an environment supports a particular process, it must be able to accommodate such changes.

Because many changes are not made in isolation but have side effects, propagation of changes must be managed. In many cases, propagation of change takes the form of posting a notice (e.g., new version of Ada compiler has been installed overnight) and performing a complete rebuild of the system—a costly operation for large systems. Techniques such as *smart recompilation* techniques [15] and notions of upward compatibility and user-specified equivalence to temporarily permit inconsistency [10, 12] reduce the cost of rebuilding a system. For large systems, it is sometimes desirable to have two versions of a component coexist in a system during transition periods. In interactive environments, the cost of immediate propagation may become high for large systems even when smart recompilation techniques are applied. Rational, for example, has introduced the notion of *subsystem*, a facility to group a set of Ada packages and act as a firewall for propagation [1]. In general, it is not always desirable for the person causing the initial change to be responsible for all side effects. The responsibility must be determined and the appropriate agent notified. Tools that have access to relevant dependency information can use this information not only to help reflect a change through automatic application of derivation tools but also to help determine an appropriate strategy for managing a set of changes through dynamic partitioning of the system into units with independent clusters of changes.

In IPSEs, change is not limited to the product. Personnel may leave the project, or a new computer system may be installed and become available as resource. As environments with integrated project management and development support are starting to manage more project-related nonproduct information, it becomes more crucial for them to provide appropriate high-level operations to deal with the effects of change to project resources, organizational structures, and information flow.

3.5. Adaptation of the Environment

An environment must be adapted to an organization before it can be successfully used on a project. One aspect of adaptation, tailoring of the processes supported by the environment, has been discussed earlier. A second aspect of adaptation is the ability to support the integration of tools into the environment. Most organizations already have tools in active use, and replacing them with a new environment is both risky and costly. In some cases existing tools must be used, e.g., a corporate accounting system or cost estimation tools tuned to the organization. Transition of a new environment into an organization can be eased if tools in active use can be made available in this environment.

Another aspect of adaptation is the use of the environment with existing products and projects. A software system may have been produced before the environment existed or in a different environment. Therefore, in order to avoid limiting its use to development of new products, facilities must be provided to import software into the environment. Importing software may consist of: bringing in source code and reconstructing executables; bringing in both source code and executables; importing requirements, design documents, and user documentation; and importing multiple versions and configurations of the above. Bringing a new environment into an ongoing project would mean that project management information must be brought into the environment as well, either by integrating the existing project management tools or by importing the information.

Finally, ease of learning an environment has to be considered as an aspect of adapting the environment. As environments grow to support more of the software life cycle as well as management of the software development, they increase the cost of training. Learning falls into two components: learning a method, and learning about the particular facilities in the environment to support the method. The latter is directly affected by the choice of user interface technology used in an environment. Availability of pointing devices, menus, windows, simplicity of concepts, and consistency of functionality affect learnability and ease of use. User familiarity with a method being supported has a greater impact on learnability. Users can learn the commands of a graphics editor or the syntax of a programming language like Ada in a relatively short time. However, learning the methods embedded in such tools takes much longer [5].

4. Conclusions

Management and control of development is perceived to be a key to the successful completion of very large software system projects. Management consists of management of the evolving product, management of the development process, management of the resources involved in the development, and management of the development environment. Tools exist for many of the management aspects (e.g., error report tracking, scheduling, version control). These tools, which were developed in isolation, are often independent of each other. In the past, humans provided the link between them and were responsible for their consistent application. Recently, integrated environments have appeared (some under the name of integrated project support environments) and attempt to address some of the management issues on a more global scale. The success of these environments in assisting with the management of very large systems will need to be proven.

In this paper, we have discussed a project scenario that addresses many of the management issues. This scenario is part of an environment evaluation methodology that has been developed at the Software Engineering Institute and has been applied to several environments. From this work, we have derived several implications that integration of project and development management support has for environment architectures. The implications highlighted in this paper are:

- Focus on user activities: Tools exist to support parts of the management of a project. By focusing on small pieces, they tend to provide operations at a different level of abstraction. Integration of management tools can raise the level of abstraction to that of the user and reduce the increase of complexity when tools based on different concepts are merged.
- Integration of project management and development support: Integration means that development support techniques, such as version control, can be applied to project management and that management support mechanisms, such as task management, can be provided in the development support facilities.
- Process modeling and automation: In environments that consist of a collection of tools, users have to adhere to conventions and procedures to consistently proceed in a project. An integrated environment has the potential of capturing some of the processes. It can then support and automate them.
- Support for evolution and change: Version control and configuration management facilities have been provided to support evolution of software. However, during the lifetime of a project, change to system descriptions, to the tools in the environment, and to the processes must be accommodated.
- Adaptation of the environment: As an environment captures more of the development process, it seems to impose more of its own view. In order for an integrated environment to be transitioned successfully, it must be able to deal with adaptation. Adaptation ranges from integrating existing tools and adapting the processes, to applying the environment to an existing product or project.

In summary, integrated project support environments that cover a full spectrum of management activities in an integrated manner will need further attention of researchers before they become reality for practical use. Environments need to take into account management of the product as well as management of activities, and to provide support facilities that reflect the evolutionary nature of software development (i.e., change to software components, software system structures, processes, plans and schedules, and the development environment).

References

[1] Archer, James E., Jr., and Devlin, Michael T.
Rational's Experience Using Ada for Very Large Systems.
In *First International Conference on Ada Programming Language Applications for the NASA Space Station*, pages B.2.5.1-B.2.5.11. Houston, TX, June, 1986.

[2] Dowson, M.
ISTAR - An Integrated Project Support Environment.
In *2nd ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments*, pages 27-33. ACM, December, 1986.

[3] Dowson, M.
ISTAR and the Contractual Approach.
In *Ninth International Conference on Software Engineering*, pages 287-288. IEEE and other computer societies, IEEE Computer Society Press, March, 1987.

[4] Feldman, S. I.
Make—A Program for Maintaining Computer Programs.
Software—Practice & Experience 9(4):255-265, April, 1979.

[5] Foreman, J., and Goodenough, J.
Ada Adoption Handbook: A Program Manager's Guide.
Technical Report CMU/SEI-87-TR-9, Software Engineering Institute, May, 1987.

[6] Hall, Anthony.
Tool Interfaces in Integrated Project Support Environments.
In *Ninth International Conference on Software Engineering*, pages 289-290. IEEE and other computer societies, IEEE Computer Society Press, March, 1987.

[7] Kaiser, Gail E., and Feiler, Peter H.
Intelligent Assistance Without Artificial Intelligence.
In *Thirty-Second IEEE Computer Society International Conference*, pages 236-241. San Francisco, CA, February, 1987.

[8] Kaiser, Gail E., and Feiler, Peter H.
An Architecture for Intelligent Assistance in Software Development.
In *9th International Conference on Software Engineering*, pages 80-88. Monterey, CA, March, 1987.

[9] Leblang, David B., and Chase, Robert P., Jr.
Computer-Aided Software Engineering in a Distributed Workstation Environment.
In *SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 104-112. Pittsburgh, PA, April, 1984.
Proceedings published as *SIGPLAN Notices*, 19(5), May, 1984.

[10] Leblang, David B., and McLean, Gordon D., Jr.
Configuration Management for Large-Scale Software Development Efforts.
In *GTE Workshop on Software Engineering Environments for Programming in the Large*, pages 122-127. June, 1985.

[11] Minsky, Naftaly H.
Controlling the Evolution of Large Scale Software Systems.
In *Conference on Software Maintenance-1985*, pages 50-58. November, 1985.

[12] Narayanaswamy, K., and Scacchi, W.
Maintaining Configurations of Evolving Software Systems.
IEEE Transactions on Software Engineering SE-13(3):324-334, March, 1987.

[13] Ramamoorthy, C. V., Garg, Vijay, and Prakash, Atul.
Programming in the Large.
IEEE Transactions on Software Engineering SE-12(7):769-783, July, 1986.

[14] Tichy, Walter F.
RCS—A System for Version Control.
Software—Practice and Experience 15(7):637-654, July, 1985.

[15] Tichy, Walter F.
Smart Recompilation.
ACM Transactions on Programming Languages and Systems 8(3):273-291, July, 1986.

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

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

[18] Weiderman, N. H., and Borger, Mark H.
Generic Evaluation Experiments for Assessing an Ada Environment's Support of Configuration Management Activities.
In *Proceedings of the Ada Europe Conference, Stockholm, Sweden*. May, 1987.

Table of Contents

List of Figures