DEC TP WORKcenter: A Software Process Case Study

by

Ernesto Guerrieri and Bruce J. Taylor

ABSTRACT

DEC TP WORKcenter is Digital's object-based production system development environment for Application Control and Management System TP applications. Goals for the DEC TP WORKcenter project were to meet customers' requirements, to provide superior product quality, and to maintain schedule predictability. Modern software process techniques helped to achieve an appropriate balance in resolving the inevitable conflicts between project goals. A critical analysis of each software process shows its effect on the engineering team, the product, and the project schedule. Changes to the process were implemented based on the team's experience and quality metrics. Recommendations to other project teams are offered based on the conclusions drawn from the DEC TP WORKcenter project.

INTRODUCTION

The DEC TP WORKcenter product is an interactive production system application development environment specifically customized for Application Control and Management System (ACMS) transaction processing (TP) applications.[1] Development of the DEC TP WORKcenter object-based development environment started in 1991 in response to requests from a number of Digital's ACMS customers. They wanted a tool that could help them to

o Perform configuration management of ACMS application components

o Track ACMS application components

o Obtain a more efficient build mechanism for ACMS applications

The product development team consisted of a team leader, an architect, six software engineers, a quality engineer, two test engineers, and two documentation writers. The average experience of the team was seven to eight years of industrial experience (with at least three members having over ten years of experience) in a wide variety of software industries, including defense-oriented developments. This breadth of experience was important in the creation and adoption of the development process.

The key goals of the project were to provide

- o Customer-defined product requirements
- o Compliance with the product requirements specification
- o A high-quality product
- o Delivery on schedule

For the customer satisfaction goal, we describe our use of Contextual Inquiry, Quality Function Deployment, conceptual modeling, and rapid prototyping. We also describe a formal requirements documentation technique to analyze requirements and guide later software phases.

For the quality goal, we describe the use of the requirements document, the interface and design review process, and the use of inspections. We mention functional testing as guided by the requirements document.

For the schedule goal, we discuss the organization of the team into working groups and the use of the requirements document to ensure coverage of a requirements matrix.

Finally, we describe several management processes for balancing conflicting goals and assessing project dependencies and risks through process metrics. From this experience, we have formulated a collection of recommendations that we feel are true not only for the DEC TP WORKcenter project but for all projects.

THEME

Every engineer on the DEC TP WORKcenter development team had experience with formal or semiformal software development processes. The positive experiences came from projects that were developed smoothly and without incident. The negative experiences stemmed from projects that ended in disaster in spite of (or because of) formal development methodologies. The entire engineering team, however, was enthusiastic about formal policies, as long as the team could be in control of the process. The team's unofficial motto was

> "Use the process, but
> don't let the process use you."

Throughout the development cycle, we looked for formal techniques to control various parts of our work, and then tried to adapt these techniques to the particular requirements and capabilities of our development team. In some instances, we were able to install a formal mechanism with little or no modification; but for most cases, we had to refine the mechanism, using the following steps.

1. Document the mechanism.

2. Test it on a realistic sample task.

3. Collect objective measures of how well it worked.

4. Adapt the mechanism.

5. Repeat until satisfied.

We never used complex metrics, software physics, or deep analysis; the key to any success was to keep the process simple and to continually adapt it to fit the nature of the task and the team. Once we were satisfied with the process, we tried to apply it uniformly and consistently across the product development.

DESIGN REQUIREMENTS

Because the DEC TP WORKcenter product was the result of a customer-driven process, we were faced with a number of challenges, which can be categorized into the following three areas.

- o Gathering customer requirements efficiently, accurately, and objectively

- o Capturing and integrating the requirements of several customers into a single, coherent specification

- o Recording the requirements specification so that it could be used as a reference during design and testing phases

With the help of Digital's Software Engineering Technology Center (SETC), we focused on two techniques for gathering requirements: Quality Function Deployment and Contextual Inquiry. Furthermore, we utilized a formal requirements specification document to capture the results of these techniques. We also utilized prototypes to validate our understanding with the customers and documented this in another document, the DEC TP WORKcenter Conceptual Model.

Quality Function Deployment

Quality Function Deployment (QFD) is an exercise in forming consensus among team members (including customers and development partners) for identifying key requirements.[2,3] In a previous project, QFD techniques had been performed for many of the same functionalities of the DEC TP WORKcenter product. We evaluated the validity of the data and results of QFDs for that project to determine if they could be applied to the overlapping features in the DEC TP WORKcenter product. This method allowed us to take advantage of valid QFD data and results without incurring the cost of producing them.

Apart from the reuse of valid QFD results, we found QFDs to be a fairly expensive way to gather requirements. The QFD techniques involve a great deal of preparation, customer participation, and analysis. The results, however, justified the effort expended. We emerged from the QFD process with a prioritized list of requirements. For each requirement, we also identified (1) how well the current products satisfy the requirements, and (2) how well the competition satisfies the requirements.

All of these factors were expressed as numbers and could be readily ranked for importance, cost, and benefit. Once the requirements were ranked, we determined the features to be included in the product based on resources and projected market dates. These decisions were then validated by the customers who had been involved in the initial requirements gathering.

Recommendation: Reuse QFD data. Existing QFD data (either QFD input data and/or requirements resulting from the QFD) may be reused upon assessment of their validity.

Contextual Inquiry

Acting on the advice of the SETC, we used Contextual Inquiries (CIs) to gather requirements.[4,5] CIs are structured visits to selected customer sites to record exactly how the customer develops ACMS applications today, and exactly how a proposed solution could improve the customer's productivity. This technique involved a great deal of analysis and was an expensive way to gather requirements. We feel it was worth the cost because it gave us confidence in our requirements list. We were able to compare the requirements against actual customer activities to determine:

1. Those requirements on the list that would not be used by the customers

2. Those customer activities that would not be supported by the product as described in the requirements list

Both the CI and QFD techniques yielded firm, objective requirements specifications that could be compared, ranked, and further analyzed.

In retrospect, the CIs that had the most impact were the ones that were properly documented for future reference immediately after the CI visit.

Recommendation: Document Contextual Inquiry Data. In order to trace information to the CI and/or reuse its data, the CI visit needs to be formally documented.

Requirements Specification

We needed an effective way to capture and combine the product requirements into a formal specification that could be used as a benchmark for development. Several engineers on the team had a background in programming for the Department of Defense and were familiar with the DoD-STD-2167A development process.[6] These engineers convinced the team that the process is beneficial if it is simplified and streamlined.

Accordingly, the team analyzed the DoD-STD-2167A Software Requirements Specification format and modified the format to the project's needs. As a result, the team produced a requirements specification document that matched the scope of the project, reflected the background of the team members, and traced the origin of the customer requirements. The final document was 40 pages of semiformal prose and has remained current for the duration of the project.

We have used the requirements document as an important data source in later development phases. During software design, we compared design features to the requirements document to eliminate unnecessary design frills and to detect requirements that were not met. We referred to the requirements specification to develop a test suite for complete testing of all product features. To ensure the use of the requirements specification, the documentation should be kept as short as possible, as concise as possible, and as descriptive as necessary.

Recommendation: Customize the requirements specification. The level of formality of the requirements specification should reflect the purpose of the document. Furthermore, it should be as short as possible, as concise as possible, and as descriptive as necessary.

Prototypes and Conceptual Model

While we were preparing the requirements specification, we also built two prototypes of the human interface for the DEC TP WORKcenter environment. The first prototype existed only on paper as a series of Motif windows that illustrated how we imagined the main functions of the DEC TP WORKcenter would operate. We showed this paper prototype to customers, asked for their feedback, and made extensive modifications based on their reactions. We repeated this process at least three times. In retrospect, it was an expensive way to refine the interface, but it gave us confidence that we were building the correct interface to our product. This paper prototype was captured in a formal document called the DEC TP WORKcenter Conceptual Model and would later support the DEC TP WORKcenter Functional Specification and the user interface design.

To demonstrate that the product was practical and to get some initial performance results, we also constructed an executable prototype of a few product functions. This activity was valuable in demonstrating feasibility, but it had two unfortunate side effects. First, it distracted the team from the design process, which caused the schedule to slip. Second, we did not have the sense to discard the prototype after it served its purpose. The engineering prototype suddenly became the first base-level code and entered the main line of development. Eventually, we had to rewrite most of the prototype code, which was a more costly procedure than starting with a clean design. The engineering prototype can be a valuable step if it has a well-defined purpose and if it is discarded when that purpose is served.

Recommendation: Restrict prototype usage. The engineering prototype can be a valuable step in product development, if it has a well-defined purpose and if it is restricted to that purpose.

DESIGN PHASE

We used several techniques during the design phase, including

- o Feature-based working groups
- o Electronic design notebook
- o Layered approach to object-oriented design
- o Detail-level design header files

The feature-based working groups allowed the team to develop the high-level design in parallel in a concentrated period of time. The output of each feature-based working group was kept in an electronic design notebook and formed the evolving high-level design. Once the high-level design was completed, the team reviewed the design to validate consistency and integrity to product requirements and between interacting or dependent product features.

A layered approach to the object model was used to describe the design of the product. The layered approach allowed for easy separation of the object-oriented design from the object-oriented features of the product. After the high-level design was completed, header files were used to define the detail design of the product.

Feature-based Working Group Technique

During the design phase, we defined the major features of the product and determined which requirements affected which feature. We then formed feature-based working groups (FBWGs) to develop the design of each feature with respect to its associated product requirements. Team members participated in the FBWG of interest to them, and a designated responsible individual (DRI) led each FBWG. Since the number of team members was less than the number of working groups, team members participated in more than one FBWG. There were approximately twice as many features as there were team members. Consequently, each team member was a DRI of approximately two FBWGs and participated as a member of approximately six other FBWGs. Once membership of the various FBWGs was established, the FBWGs met, depending upon the availability of the members. Meeting conflicts were avoided by tracking FBWG meetings on a white board.

Table 1 illustrates the team members' participation in the various FBWGs for the DEC TP WORKcenter project. The columns in Table 1 represent the various FBWGs, and the rows represent the project team members. The entries in the table indicate the role that a specific team member played in the specific FBWG. The load column indicates the overall role (number of FBWG DRI roles, number of FBWG member roles) the team member played across all FBWGs.

Table 1 Feature-based Working Group Matrix

Team Member	Load D/P	WG 1	WG 2	WG 3	WG 4	WG 5	WG 6	WG 7	WG 8	WG 9	WG 10	WG 11	WG 12	WG 13	WG 14	WG 15	WG 16	WG 17	WG 18	WG 19
Engineer 1	1/11	P	.	.	P	D	P	P	.	P	P	P	P	.	P	.	.	.	.	P
Engineer 2	2/5	.	.	.	D	.	D	.	.	P	.	P	.	.	.	P	.	.	.	.
Engineer 3	2/8	P	.	.	.	P	P	D	.	.	.	.	D	.	.	.	.	P	P	P
Engineer 4	2/9	P	P	.	.	P	.	.	.	P	D	.	P	.	P	.	.	D	P	.
Engineer 5	4/8	D	P	D	.	P	.	P	P	.	.	.	.	.	D	D	.	.	.	.
Engineer 6	1/9	.	P	.	P	P	P	.	.	.	P	P	.	D	.	P	P	.	.	.
Engineer 7	1/2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	P	.	.	D
Engineer 8	3/4	.	.	.	.	.	.	.	D	.	.	D	.	.	.	.	.	P	D	.
Engineer 9	2/7	P	D	.	.	.	.	.	P	D	P	.	P	P	.	.	.	.	.	.
Documentation Writer 1	0/1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	P	.	.	.
Documentation Writer 2	1/3	.	P	.	.	.	.	.	.	.	P	.	.	.	.	.	D	.	.	.

Notes:

D -- Designated responsible individual for the WG
P -- Participant in the WG

Dependencies or interactions between product features needed to be managed. If a team member's participation overlapped with the interacting features, that person provided a means of communicating among the associated FBWGs. Otherwise, the corresponding DRIs provided this exchange of information. Also, the project leader and the architect attempted to attend all meetings to guarantee consistency across the various FBWGs. This allowed us to resolve many issues consistently, but we would have benefited from a more formal mechanism for settling design disputes.

The FBWGs continued to a lesser extent during the detail-level design, but the issues were narrower in nature and were dealt with by the FBWG DRI and the affected component DRIs.

In conclusion, the FBWGs provided clear assignment of responsibility and guaranteed that the design was covered by more than one team member. Due to their parallel nature, the FBWGs had no adverse affect on the schedule. Unfortunately, even for small groups, the FBWG generated too much specialization of knowledge.

Recommendation: Adapt the design process. The design process should be adapted to meet the schedule and resource constraints.

Electronic Project Notebook

The minutes and draft/final design of each FBWG were recorded in an electronic project notebook. The electronic project notebook provided a means of communicating the evolving design of the product among the team members. Once entered into the notebook, the information was made available to the team. Also, the entries posted in the notebook during the day were collected and mailed electronically to the team members every night so that the team remained current on all design issues and decisions. This proved an efficient method for communicating the information to the entire team as well as for recording the information for later use.

Without a goal to produce a formal design document, the team members were not as careful in documenting their design. Furthermore, the design was dispersed over a set of notebook entries that created issues in two areas:

- o Configuration management: Which notes formed the current set of design notes?
- o Inspection difficulty: Which version of a design note was a source document?

The electronic project notebook was not limited to the design phase but was used to record and exchange information throughout the phases of the product development life cycle.

Recommendation: Capture project information. The electronic project notebook is an easy way to share knowledge and exchange ideas, issues, solutions, futures, etc., about a project.

Recommendation: Generate formal design specifications. Although the electronic project notebook contained the design, it is not a substitute for a formal design specification.

Layered Approach to Object-oriented Design

Since the product would be object-based, we used object-oriented design (OOD) techniques. Due to the inexperience of some team members, the distinction between abstraction levels was not always clear. To allow the team to recognize the different abstraction levels, we used different languages for the two levels of abstraction. At the product level, object-oriented terminology was used. At the product architecture level, a constrained layered model was used in which the constraints allowed a simple mapping into an object-oriented model.

The following constraints were applied to the various layers in the model.

1. Each layer provides one and only one specific type of resource.

2. Each layer provides a set of services to manipulate that resource.

3. The resource and/or its services may use other layers to provide needed resources and services.

These rules allowed the team to distinguish between the design of the product and the data model of the objects manipulated by both the product and its object-based operations. Although this layered approach to OOD was formulated to make use of the team's background, the resulting design was not a pure OOD.

Recommendation: Understand the purpose for modifying a process. Although the layered approach to OOD attempts to bridge traditional design methods to OOD methods, it should represent only a phase in a planned transition to OOD techniques.

Detail-level Design Header Files

During the detail-level design stage, we refined the various layers required to implement the resources and services to support the product features. This included determining the final interface of each layer, defining the resource controlled by the layer, and describing the functionality of the services provided by each layer.

To optimize consistency and effort, the detail-level design was represented as a C header file that provides the services of a layer implemented in a C module. Furthermore, if a module represents an object, then the header file consists of the visible operations that can be performed on the object.

The header files were placed under configuration control while issues and resolutions concerning a layer were recorded in the electronic design notebook.

Since several features required the services of a specific layer (later implemented as a C module or component), we captured the relationships in a feature/component matrix. Table 2 gives the feature/component matrix for the DEC TP WORKcenter product. The columns in Table 2 indicate the various product features, and the rows indicate the components of the product. An entry in the matrix indicates that the component implements or supports part of the product feature.

Table 2 Feature/Component Matrix

Compo-	Features																					
nents	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.
2	.	.	3	3	3	3	3	3	.	3	.	3	.	3	.	.	.	.	.	.	.	.
3	.	.	3	3	3	3	3	3	.	3	.	3	.	3	.	.	.	.	3	.	.	.
4	.	2	2	1+	2+	3	3	2+	.	3	2+	3	2	.	.	.	2+	.	.	.	.	D
5	.	3	3	3	3	3	3	3	.	3	3	3	3	.	.	.	.	.	3	.	.	D
6	.	2	2	2+	2+	3	3	2+	.	3	2+	3	2	3	.	.	.	.	.	.	.	D
7	.	2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2+	.	.	.	.	.
8	.	.	2	2+	2+	3	3	2+	.	3	2+	3	.	.	.	.	.	.	.	.	.	D
9	.	.	.	.	.	.	.	.	.	3	2+	.	.	.	.	.	.	.	.	.	.	.
10	.	.	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.
11	.	.	.	.	.	.	.	.	2+	.	.	.	.	.	.	.	.	.	.	.	.	.
12	.	.	.	.	.	.	.	.	2+	.	.	.	.	.	.	.	.	.	.	.	.	.
13	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	2	.	.	.	.	.
14	.	.	1+	1+	2+	3	3	2+	2	3	2	.	.	.	.	.	2+	.	.	.	.	.
15	2	.	1+	1+	2+	3	3	2+	2+	3	2	.	.	.	.	.	2+	.	.	.	.	.
16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
17	.	.	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.
18	.	.	.	.	.	.	.	.	.	.	.	.	1+	.	.	.	2+	.	.	.	.	.
19	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1+	.	.	.
20	.	.	.	.	.	.	.	.	.	.	.	.	1+	.	.	.	.	.	.	.	.	.
21	2	.	.	.	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
22	2+	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2+	.	2+	.	.
23	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2+	.	.
24	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.	3	.	3	.

Notes:

1	: Base Level 1
1+	: Base Levels 1 and 2
2	: Base Level 2
2+	: Base Levels 2 and 3
3	: Base Level 3
D	: Deferred

A DRI was assigned to each header file to coordinate the needs of the various features on that layer. The component DRI met with several FBWG DRIs to ascertain the needs of each feature and present a satisfactory interface. On the other hand, each FBWG DRI needed to coordinate with several component DRIs to provide the capability for the associated feature.

Recommendation: Share information across development phases. The use of header files as part of the detail-level design provided (1) a centralized location for all interface information about a module, (2) no redundancy of interface information, and (3) an up-to-date interface in the corresponding code.

Design Reviews

The entire team reviewed the high-level design for consistency across the various product features and for integrity of the dependencies between features. Due to time constraints and the amount of design information, this review was inefficient and was not formally completed. Marathon high-level design review did not work since it was too intense and too long. We concluded that the review process must be streamlined.

The detail-level design was represented as C header files. Consequently, they were targets for code inspection.

Recommendation: Review the design in manageable pieces. Divide the high-level design into modules so that its review is manageable.

CODE INSPECTIONS

Although inspections were used for the requirements document and the data model design, most of the inspections occurred during the DEC TP WORKcenter coding phase. The technique was modified to deal with time constraints and the amount of coding, and to gain the acceptance of the team on the usefulness of inspections. Basically, we defined a formal inspection and a semiformal inspection.

The formal inspections follow the guidelines as described by Fagan.[7,8] The semiformal inspections had the following restrictions:

1. Only two engineers participated in the inspection.

2. The moderator was also the reader.

3. The author was also the recorder.

The following criteria were established to decide which type of inspection would be performed.

1. Complex code was formally inspected.

2. Critical code was formally inspected.

3. Remaining code was informally inspected.

The complexity of the module was determined by computing the McCabe cyclomatic complexity of the module.[9,10] The threshold for complex code was initially set at 7 and would be periodically adjusted based on feedback on the effectiveness of the inspections. Note that the literature has usually determined 10 to be this threshold. At 7, approximately 17 percent of the code was considered complex. This may be attributed to either the tendency of modules to represent objects in the design or the use of the X Window System and Motif as the graphical user interface.

The project leader determined the critical code according to the nature of the code or intermodule dependencies in the system. This information was available from the detail-level design. One example is DEC TP WORKcenter parsers, where the flow of control is based on pattern triggers rather than on sequential execution of statements. Consequently, the DEC TP WORKcenter parsers were deemed to be complex.

All remaining code was inspected using semiformal techniques. To discourage the engineers from artificially constraining their code to be noncomplex, the project leader could randomly choose code for formal inspections (this was never needed).

As another refinement to the inspection process, we reduced and adapted the set of codes used to characterize a defect according to the type of document being inspected. This technique allowed us to accelerate the inspection and continue to capture the information of interest.

In another attempt to refine the inspection process, the recorder defined the defect codes. This accelerated the semiformal inspections but slowed the formal inspections.

Recommendation: Understand the purpose for modifying a process (revisited). Under schedule or resource constraints, consciously decide how to formally relax the inspection process and understand the consequences.

Recommendation: Choose tools to support the process. Given unbiased criteria to select the level of inspection, choose the appropriate tools to support the decision process.

SCHEDULING

Project scheduling played an important role in managing the project. Scheduling tools associated with personal computers (such as program evaluation and review technique [PERT], critical path method [CPM], precedence network, and resource leveling capabilities) were used to manage the schedule. Tasks were classified as either process-related or product-feature-related. The process-related tasks covered activities such as Digital's Phase Review Process or customer interactions. The product-feature-related tasks were activities directly related to the design, implementation, and testing of product features.

One distinction of the DEC TP WORKcenter product is that most of the product-feature-related schedule was determined from the feature/component matrix (see Table 2). When a specific feature was planned to be added into the product, the components supporting that feature were also scheduled to be added. The entries in the matrix in Table 2 indicate in which code base level the component implements or supports the product feature.

The engineer(s) assigned to a task submitted an estimate of the time needed to accomplish the task to the project management. If the estimates were considered unreasonable based on past engineering experiences, an in-depth analysis was performed to understand the discrepancy. These discrepancies were due to either a misunderstanding by the project management of the complexity of the task or an inefficient solution plan by the engineer to build upon existing components or processes.

Recommendation: Share information across development phases (revisited). Use requirements analysis and design information to define the schedule.

Recommendation: Get team support for the schedule. For any schedule, obtain commitment from the team.

Efficiency Factor

We also calculated an efficiency factor to account for activities that would lower the efficiency of engineers in performing their tasks. These activities included periodic mail reading, attending non-project-related meetings, sick time, jury duty, and code inspections. We revised all work estimates to reflect the engineer's efficiency factor. Initially, the efficiency factor for most of the engineers was calculated to be 60 percent. Although the efficiency factor was intended to achieve the most realistic schedule possible, it was the cause of several problems:

- The efficiency-related activities were counted twice if the engineer's estimates included these activities.

- There is an assumption that the efficiency-related activities are spread uniformly over all tasks. This is true for repetitive activities that occurred within the resolution of the tasks being estimated, but other efficiency-related activities occurred rarely (e.g., sick time) or were associated with a specific phase of the project (e.g., code inspections).

As a result, the schedules needed to be refined and adjusted frequently.

Recommendation: Understand the factors that impact the schedule. The efficiency factor attempts to capture those separate activities that were not worthwhile but impact the efficiency of other activities.

Unplanned Tasks

During the initial phase of the project, the project management recognized that schedule predictability was highly influenced by unplanned tasks. To better understand the nature of unplanned tasks, the project management participated in a Software Metrics In Action (SMIA) course offered by the SETC. The SMIA course was applied to our problem of unplanned tasks over the next phase of the project. To our surprise, we concluded that, no matter how well one plans, one always has an additional 20 to 25 percent of unplanned tasks. This included new tasks, existing tasks that took longer, and existing tasks that were completed.

Recommendation: Understand the impact of unplanned activities. No matter how well one plans, one always has an additional 20 to 25 percent of unplanned tasks. This includes new tasks, existing tasks that took longer, and existing tasks that were completed.

Milestones

The difficulties of estimating tasks and the existence of unplanned tasks would sometimes render the schedule invalid.

Milestones within the project schedule allowed the team to meet the associated deadlines. Milestones also caused two events that affected the project:

- o Unplanned tasks were prioritized against planned tasks, causing readjustment of milestones based on the prioritization criteria.

- o Engineers became more efficient, causing the efficiency rating to be revised and allowing some of the unplanned tasks to be included without impacting the schedule.

Recommendation: Define milestones. The team works best when well-defined milestones for goals are established.

Feature "Hit List"

Toward the end of the design phase, we determined that the planned date for completion could not be met unless we reduced the functionality of the product. We created a feature "hit list" in the electronic project notebook in which we listed the candidates for elimination from the product. The feature hit list was used in a Pugh process to determine, in a structured manner and with group consensus, the features to be eliminated in order to meet the projected market date.[11]

Some of the features that we eliminated through our hit-list technique originated in the QFD process. During field test training, customer feedback indicated that some of the eliminated features were needed for a viable product. This event caused us to reevaluate and readjust the projected market date in order to include the missing features. Thus, we reaffirmed the validity of the results supporting our customer satisfaction goal.

Furthermore, the readjustment of the projected market date had high management visibility, but the utilization of the customer satisfaction processes permitted us to adequately document the rationale for and justification of the readjustment.

Recommendation: Manage and adapt the change process. When making a change that is visible to the customer and/or management, one needs (1) a formal process for defining the change, (2) consensus among the team, (3) traceability to facts supporting the original decision and its change, (4) impact analysis of change, and (5) agreement from customer and/or management.

FINAL PHASE

In the final stages of the DEC TP WORKcenter product development, we conducted field tests at customer sites, identified defects, and determined the final changes to be made to the product.

Field Test Advocacy Program

During field test, we took a proactive approach in our relationship with the customer field test sites. Under our Field Test Advocacy Program, an engineer is assigned to monitor the progress and to resolve any issues or problems at the customer's field test site. The engineer monitors the customer's software problem reports (SPRs) in the field test SPR database to understand (or be aware of) any patterns in SPRs.

In one example, a customer raised a series of feature suggestions that were all attempts to use the DEC TP WORKcenter environment for an unsupported object type. Although the suggested features would be useful, they would not be as important if the main feature was provided. Monitoring customer SPRs provided us with an understanding of how the customer was testing and assured the customer that the engineering team understood the customer's concerns.

Recommendation: Adopt useful processes. Adopt processes in which the benefits outweigh the costs, but understand the time frame of both.

Tracking Defects and Monitoring Fixes

As the product was being developed, all (internal and external) problems were tracked using a problem tracking tool. Every problem was entered into the problem database and given a unique identifier. This allowed the engineer to associate a fix with the corresponding problem identifier. Furthermore, the problem tracking tool allowed us to monitor the defect identification and fix rate on the project. Figure 1 shows both the number of problems entered over time as well as the problems fixed over time.[12] Interesting points in the graph are the slopes, plateaus, change in slope, and vertical distance between the two lines.

The tracking tool also allowed us to verify that the priority of the fixes was consistent to the severity of the problem. Figure 2 shows the same graph for the two highest severity classes and indicates that the problems with the highest severity classes were monitored closely and fixed immediately.

Tracking the problems worked well to identify issues during the DEC TP WORKcenter product development. More analysis, however, was needed to understand trends as soon as possible.

Recommendation: Adopt processes to collect valuable metrics.

Understand the rationale for adopting a metric and set up a process that achieves the goal of the metric.

MUST-DO Lists

As we approached major code freeze dates, we prioritized the defects to be fixed and compared them to our MUST-DO criteria. Usually the criteria consisted of the following.

- The defect was a priority 1 or 2.
- The defect impeded testing efforts of critical functionality.
- The defect represented a regression from a previous stable version of the product.

The defects were added to the MUST-DO list if they met the criteria. This list indicated backlogs of defects that needed to be resolved prior to declaring a code freeze. Figures 3 and 4 show MUST-DO count patterns prior to reaching code freeze. The solid line (total) indicates the outstanding MUST-DO items over time.

Recommendation: Define valuable metrics (or focus on important issues). The MUST-DO list helps prioritize the tasks that require focus during a specific activity and provide well-defined goals for the team.

Product Stability

Once the product had reached feature freeze, a change control board was put in place to guarantee the stability of the product and to avoid any major regression that would impact the schedule. The board approved the inclusion of any defect fix after (1) review or inspection of the code modifications, and (2) adequate testing.

Furthermore, we monitored the defect discovery rate to determine if it was stable enough to warrant a code freeze.[12] In this case, we measured a running total of the number of MUST-DO items added over the last five days. Figures 3 and 4 show this metric. The broken line (five-day cumulative) indicates the five-day running total and measures if the changes are stabilizing.

Recommendation: One can always improve. It is never too late to set up a change control board to reduce the introduction of new problems and regressions.

CONCLUSIONS

The DEC TP WORKcenter object-based development environment (version 1) was developed over approximately 24 months. During this time, we were presented with a variety of situations that could have impacted our project goals. This paper presents several of the processes that the team adopted to meet the project goals. Table 3 summarizes the recommendations based on our experiences on adopting processes to support our goals. In retrospect, we see that the project functioned smoothly when all of the following conditions were met.

- Everyone understood what development phase was in progress.

- We identified a set of processes to govern each phase.

- We adapted the process to suit the project team.

- We adapted the process to the realities of the project schedule.

- All the team members understood and accepted the process.

- We followed the process conscientiously.

In short, the entire experience of the DEC TP WORKcenter project can be summed up as:

- Software development processes should be as simple as possible.

- The team should formally adapt the processes to its own needs.

- The team should understand the consequences of modifying the process.

Although these rules of thumb do not ensure a smooth, productive project, the DEC TP WORKcenter team found them to contribute to a successful conclusion.

Our recommendations can be adopted by any project team; however, the team would benefit by taking part in a similar process of identifying its goals and supporting them with appropriate processes.

Table 3 Recommendations Based on the DEC TP WORKcenter Development Project

1. Reuse QFD data.

2. Document Contextual Inquiry data.

3. Customize requirements specification.
4. Restrict prototype usage.
5. Adapt the design process.
6. Capture project information.
7. Generate formal design specification.
8. Understand the purpose for modifying a process.
9. Share information across development phases.
10. Review design in manageable pieces.
11. Choose tools to support process.
12. Get team support for the schedule.
13. Understand the factors that impact the schedule.
14. Understand the impact of unplanned activities.
15. Define milestones.
16. Manage and adapt the change process.
17. Adopt useful processes.
18. Adopt processes to collect valuable metrics.
19. Define valuable metrics (or focus on important issues).
20. One can always improve.

ACKNOWLEDGMENTS

The authors would like to acknowledge the members of the DEC TP WORKcenter team, past and present, that helped define and adopt the various processes presented in this paper. Also, the detailed comments from the many reviewers were very helpful.

REFERENCES

1. T. Speer and M. Storm, "Digital's Transaction Processing Monitors," Digital Technical Journal, vol. 3, no. 1 (Winter 1991): 18-32.

2. J. Hauser and D. Clausing, "The House of Quality," Harvard Business Review, vol. 66, no. 3 (May-June 1988): 63-73.

3. L. Cohen, "Quality Function Deployment: An Application Perspective from Digital Equipment Corporation," National Productivity Review, vol. 7, no. 3 (Summer 1988): 197-208.

4. K. Holtzblatt and S. Jones, "Contextual Inquiry: Principles and Practice," Technical Report DEC-TR 729 (Maynard, MA: Digital Equipment Corporation, 1990).

5. D. Wixon, K. Holtzblatt, and S. Knox, "Contextual Design: An Emergent View of System Design," Technical Report DEC-TR 686 (Maynard, MA: Digital Equipment Corporation, 1990).

6. "Military Standard Defense System Software Development," Technical Report DoD-STD-2167A (Washington, D.C.: U.S. Department of Defense, 1988).

7. M. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," IBM Systems Journal, vol. 15, no. 3 (1976): 219-248.

8. M. Fagan, "Advances in Software Inspections," IEEE Transactions on Software Engineering, vol. SE-12, no. 7 (July 1986): 744-751.

9. T. McCabe, "A Software Complexity Measure," IEEE Transactions on Software Engineering, vol. SE-2, no. 6 (December 1976): 308-320.

10. T. McCabe and C. Butler, "Design Complexity Measurement and Testing," Communications of the ACM, vol. 32, no. 12 (December 1989): 1415-1425.

11. S. Pugh, Total Design: Integrated Methods for Successful Product Engineering (Reading, MA: Addison-Wesley, 1991).

12. R. Grady, Practical Software Metrics for Project Management and Process Improvement (Englewood Cliffs, NJ: Prentice-Hall, 1992).

TRADEMARKS

BIOGRAPHIES

Ernesto Guerrieri Ernesto Guerrieri, a principal software engineer in the Production Systems Group, is the TP WORKcenter project leader. He is an adjunct professor at Boston University. Prior to joining Digital in 1990, he was employed by SofTech, Inc., where he was the chief designer and developer of reusable Ada products for information systems development (RAPID) center library. He holds a Ph.D. in software engineering from Rensselaer Polytechnic Institute and an M.S.C.S. from the University of Pisa. Ernesto has published various papers in software engineering. He is a member of ACM, IEEE Computer Society, and Sigma Xi.

Bruce J. Taylor A principal software engineer, Bruce Taylor is the software architect of the TP WORKcenter project. Prior to joining Digital in 1991, Bruce worked in CASE tool development at Intermetrics, Inc. He designed the repository database for the SLCSE software development environment and has published many papers on the use of database technology in the software development environment. He has a B.A. in English and an M.A. in computer science from Duke University. His current research interests include repository support for complete software life-cycle environments and software quality strategies.