TRAINING: PLUGGING THE INFORMATION GAPS

By Carol J. Scott

Jet Propulsion Laboratory, California Institute of Technology
Pasadena, California, USA

ABSTRACT

Training is commonly viewed as an add-on function to the development cycle. It is imperative that this view be changed. The training developer needs to be a colleague in the system development process, contributing and learning along with the other development participants.

Training developers can make contributions to design concepts that favor end users. Early involvement will enhance the likelihood of training availability concurrent with software delivery. End users will benefit and cost-savings will be realized.

1. INTRODUCTION

As software training engineers responsible for workstation course design and implementation, we have an obligation to provide our customers with information and learning experiences that translate directly to their working environment. "Customer" is the key word, and, at the Jet Propulsion Laboratory, our customers are the flight projects. They require a supportive, well-informed, educational resource.

Training will be a more effective resource when we are active participants in project development activities. Without interaction in system development, the thought processes that justify the design are lost to us and we have a lack of perspective on resulting software products. Many design and interface decisions are made without our representation, decisions that will directly affect training quality. Training engineers are overlooked in the early phases of software development because of the lack of a "visible need" for our attendance at the design table. Funding sources look for immediate evidence of their working dollars, and anticipated productivity for training is low during the development phase.

Training may be one of the last steps on the "development ladder," but it is the first step, and often the most important, for end users.

2. THE TRADITIONAL SOFTWARE DEVELOPMENT CYCLE

The traditional software development cycle begins with a customer's need, formally specified by system engineering in a Functional Requirements Document (FRD). The FRD is delivered to software specialists, who interpret the requirements and make them a functional reality in the form of a software program.

The completed software is delivered for testing, where it must conform to the requirements of the FRD . Testers flag performance failures and the software is "fixed" and redelivered by the Software Developers. The "accepted" version is delivered to the customer, where system administration personnel perform software installation procedures for operational end users.[1]

Training developers, who at this point are end users, jump into action to learn how to operate the new software. We also incorporate input from operations personnel on end user tasks to analyze and understand how the software will be used on the job.[2] Only then can we develop appropriate learning modules and training materials. Training development often exists as an isolated effort which interfaces to the remnants of a development task force who are no longer interested in rehashing the "why's" of its design.

3. DISCOVER THE INFORMATION GAPS

The traditional positioning of training development at the end of the software development cycle limits our level of understanding to whatever comes "out" of the process. Imagine trying to train a quarterback if everything you knew about football came from sports

---

1 Delivered software is controlled by Configuration Management; operational procedures are documented and released as user guides.

2 New and modified funtional requirements will be generated by Operations as user needs change or grow.

recaps on the nightly news. You've got to be in the "locker room" to understand the plays!

3.1 The Customer/Training Information Gap

Space flight projects depend on the Training function to bring their operations and engineering personnel to a state of readiness.

3.1.1 Problems

Training is a multi-mission service provided to project operations and assumed to be available for scheduling when needed. Unfortunately, training engineers often take delivery of the software at the same time the project does and the training development effort may be just beginning. The project hardware and personnel buildup may be nearing completion so we are under pressure to commit to a training delivery schedule.

We frequently know very little about our prospective students. We often "train" new project recruits whose jobs are still undefined; sometimes we get entire teams that won't be receiving computers for months; and occasionally people are sent for training to take every class we offer because they "might need that information someday." Workstation training is an intuitive process that offers a multitude of flexible programs with capabilities and options for handling nearly any given task. The "right way" to perform a task is a subjective decision based on the user's processing goals, an existing set of variables, desired results, and user experience.

Some students may know little or nothing about their jobs. These people come to training expecting us to give them a purpose. Without a thorough understanding of mission and task goals, it is difficult for trainers to recommend a course of action in the learning environment. This comes home to us very clearly when our students pose logical "what, why, when, and how" questions that we are unable to answer.

We don't always know the environmental conditions under which user groups may be operating. Software access methods may differ from one user to the next, even on the same workstation, depending upon each user's environmental configuration characteristics. Individual teams, or positions within a team, may use a common set of programs, but have a unique perspective on the application and a different set of goals.

3.1.2 Solutions

The customer must provide training engineers with a current set of training requirements (or the FRD and an interpreter) to create a bridge between the functional requirements and the positions to be trained. This bridge will present a clearer picture of why our students are at the training table. Updates to these requirements are also needed to keep the training engineers up-to-date.

The customer is the only source of detailed information about what end users need to learn. A Position Description Document (PDD) cross-referenced to the system's functional requirements would be extremely helpful. With this information we could determine specific areas of responsibility and better estimate the level of training required. We could also provide an "educated guess" at the costs of training to that level.

Even with a PDD, a task analysis is difficult to perform for custom software users because of the enormous number of decision points to be considered. One suggestion, to make it easier, would be to use a Software Capability Checklist as a training objective selection tool for interpreting the Position Description Document. This checklist identifies and eliminates potential training objectives which are not addressed by the PDD, then reveals paths to decision points where major options can be associated with specific objectives.

The customer should identify the experience levels of prospective trainees. Classes are more productive when participants have similar knowledge and experience. A rookie is often less willing to ask questions when paired with an expert; likewise, it's difficult to maintain the interest of an expert who is pacing a rookie.

As training engineers, we need a permanent, meaningful point of contact on each project to supply us with information and make decisions for all their users.

3.2 The System Engineering/Training Information Gap

In addition to developing system functional requirements, System Engineers design operational hardware and personnel interfaces.

3.2.1 Problem

There is often a significant difference between the proposed operational working environment and what users can actually access. When there is no

consistency between the learning environment and the working environment learners have no way to apply their new skills. In these cases training can be a costly, frustrating experience.

3.2.2 Solution

The workstation environmental configuration needs to be defined and provided to training engineers prior to the start of training development. If multiple configurations will be used to accommodate the needs of different teams, it is essential that the training workstations have each configuration installed for user training.

3.3 The Software Development/Training Information Gap

Software development consists of specialists who interpret functional requirements from project system engineers, design software to those requirements, and implement the design.

3.3.1 Problems

Software developers interface directly with system engineers who help delineate developing software, but often have little, if any, communication with the users who will be operating it. When software is difficult to learn, end users may seek alternate methods to achieve their processing goals. In a distributed workstation environment, users can (and will) build their own tools to get their jobs done. These tools may propagate throughout the system, spreading by word-of-mouth, without benefit of configuration management or any kind of version control process.

Trainers occasionally have to assume the role of a marketing agent and "sell" users on the capabilities of a software product with a reputation for being too complex. This is an example of why software developers should make sure trainers thoroughly understand and "like" their programs.

Software developers do not always have direct contact with the end user community. They need insight into potential user responses under various conditions. They need someone to test drive the latest iterations of a developing program, someone with a user-like perspective who is willing to provide constructive, "in-house" feedback prior to formal testing.

Training developers do not always get intimate details about what the software is capable of doing, how it processes internally (how it "thinks"), with what subsystems it will interface, what established processing limits are, what optional processing procedures exist, etc. A trainer cannot teach the functional aspects of a program if he or she doesn't understand them. Trainers are comfortable teaching what they know well and gloss over things they don't.

Software frequently contains undisclosed program functions and built-in traits that users may discover before trainers do. These discoveries may weaken a trainer's credibility.

3.3.2 Solutions

Early access to software products would enable us to begin developing a training strategy.

Pair training developers and software developers. Trainers are often the closest thing to a marketing representative that a software developer may have. Software developers have a vested interest in ensuring that trainers completely understand the logic and functionality of their software products. Understanding the logical sequencing of algorithms will directly affect the trainers ability to answer the "why's" and "what if's" invariably asked by students. That knowledge will strengthen a trainer's credibility with learners.

Trainers also make great user advocates because they possess insight for predicting user preferences and how users will respond to software functionality under given conditions. This insight helps the software developer provide a more usable product.

Training developers must be allowed to "gorilla test" the latest iterations of the software. Early access to program operation makes it possible to develop an early training strategy. Trainers may be able to identify software bugs that could otherwise go undetected prior to testing or even operational use. Early detection provides software developers with an opportunity to "fix" them without failure report processing.

Trainers should be able to validate their developing training modules by lending their presentation skills to developers, assisting them in software performance demonstrations and design walk-throughs.

3.4 The System Administration/Training Information Gap

System administrators (SAs) work directly with users to provide and maintain a distributed system. Training developers are also users, but they rely on the SAs for additional assistance with a variety of special needs.

3.4.1 Problems

We don't know enough about system administration functions, policies, and requirements, or how to design training modules to them.

Training developers may not have access to specific information about each workstation, i.e., what software is available, which versions are installed, what remote access methods are valid, who has access, who "owns" each workstation, who maintains it, and what restrictions are currently in force.

3.4.2 Solutions

Limited cross-training would provide training developers and system administrators some perspective into how the "other half" functions, giving a broader perspective of the system.

A "who's who" map of system nodes would help orient us toward our learners, and a current workstation status report would pinpoint user capabilities.

4. A LESSON IN COMMUNICATION

Prior to the Mars Observer spacecraft launch, we were approached by the Mars Observer Spacecraft Team, which was preparing for its upcoming integration and readiness testing. They needed their engineering personnel trained in workstation operation and downlink processing as soon as possible. The hardware and software installation process was far from complete, but some of their users had workstation access.

Project deadlines were rigid and training short-cuts were required to get them on-line for downlink process testing. We shortened our list of objectives, but the limited seating in our training facility still presented a problem. With two workstations, side-by-side, we could conduct only one hands-on class at a time.

Our alternative was a "housecalls" approach. We could accommodate more students per class by conducting training in their own working environment, using their own workstations. The Spacecraft Team suggested a common area housing six workstations, where all had visibility to a white board, but they were not all within sight of each other.

We decided on the housecalls approach for our "hands-on" participation modules while holding concurrent demonstration overviews in the training facility. This got users on-line quickly during the initial phase.

The existing user environments on the six workstations determined our next step. Multi-mission Ground Data System workstations run X Windows System software under UNIX with a Motif™ user interface. Our previous customer was the Ulysses project using icon access to files, directories, and processes. We were not prepared for what we found on the Mars Observer Spacecraft Team workstations.

The six Spacecraft Team workstation environments had no common user configuration and, more important, our existing training materials did not resemble the access methods being used. Mars Observer was not using an icon-oriented desktop tool; instead, they preferred a command line interface with pop-up menus. We had presumed that all multi-mission workstation environments would be consistent in their appearance; at least that was the rumor that training received, but these workstations weren't even close.

Although we were familiar with multi-mission software components from the perspective of other projects, we knew very little about Mars Observer, the Spacecraft Team, or the needs of its engineers. All we could promise was a "best efforts" approach to the task. They agreed that workstation consistency was desired, so we went in search of some configuration support.

Training's previous concerns over a multi-mission workstation training configuration had been addressed and responsibility had recently been assigned to our Section's Operations Engineering Laboratory. They had just been incorporated into the new Operations Technology Group and were still attempting to define their responsibilities. Our problems prompted the immediate formation of the Customer Adaptation Team which became a mini-development effort to support the Mars Observer Spacecraft Team and eventually other Mars Observer organizations upon request.

The significance of the Customer Adaptation Team to the training function was our involvement. We provided the impetus for their mobilization and they included us in the development process. Using one of the six workstations as a base of operations, the Customer Adaptation Team propagated its new user configuration to our training workstations and the other five workstations in the Spacecraft Team area. We were able to make practical recommendations that benefited the users and we received each iteration of their configuration to evaluate and use in

designing our new materials. Our materials become the documentation for their development effort. The new configuration included pop-up menus and graphic user interface displays. They also provided a controlled directory structure for storing the large numbers of project-controlled scripts and required process support files that had previously been elusive entities without version protection.

Our first Spacecraft Team class was held without time for a dry run, so the first day was a debugging session. Only two users had accounts, and those accounts were not valid on all machines; some of the disks were full so that whenever we encountered a program that logged information to a file, it would "hang," etc. Without direct eye contact, it was difficult to communicate over the noise of the machines and there were numerous distractions, but at the end of the day our students' response was positive. Each day went more smoothly than the last, and we proved to ourselves that housecall training is an option worthy of serious consideration.

The most meaningful result was the obvious convenience to users. There was no transition; the learning environment became their working environment. Every workstation they were authorized to use contained the same functional configuration. *The point is that consistency is essential to reinforce learning.*

5. CONCLUSIONS

End users depend on training to lead them through the learning experience with dignity. They expect trainers to understand all aspects of the software and the processing methods necessary to accomplish and expedite their processing responsibilities. They expect trainers to translate the capabilities of the software into performance methods for accomplishing their tasks. They expect practical responses to their "how do I get from here to there" questions, even if it means providing a work-around for their specific needs.

For training to benefit the end user, it needs to be a quality and timely experience. It needs to be available when the user needs it, and the best way for that to happen is for a trainer to be a sponge for knowledge, a conductor of information, and an advocate for the end user during the development cycle. Software trainers need to experience the software, first-hand, before it hits the streets. They need to become intimately familiar with the inner workings of the software and not simply exposed to a "hand-is-quicker-than-the-eye" public demonstration.

Trainers have a unique perspective on the functionality of software in the hands of end users, and can provide insight during the design process when changes are inexpensive and easy to implement. Pairing training engineers with development engineers benefits both. Trainers gain the experience and insight into the product necessary to begin early development and testing of training materials. Programmers gain a resource with a users' point of view for dry-running early versions and providing feedback on usability. Trainers are experienced presenters and can assist in demonstrations and overviews that illustrate software development status.

6. RECOMMENDATION

Strive for a single "package delivery" for software, training, and documentation products based on functional requirements. Cross-communication provides consistency among software, documentation, training, and end user needs.

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.