Managing a Multicompany Software Development Project

The development of the Common Desktop Environment version 1.0 involved a joint engineering project between four companies that normally compete in the marketplace.

by Robert M. Miller

In March of 1993 executives from HP, IBM, Sun Microsystems and USL (now Novell) announced at a UNIFORUM conference the COSE (Common Operating Software Environment) initiative. The purpose of the initiative was to unify the UNIX® operating system industry in the areas of distributing computing (ONC, ONC+, DCE),* multimedia technology, object technology, graphics, system management, and desktop technology. While there have been other attempts to unify aspects of the UNIX world, COSE was different in that it succeeded in producing results. CDE (Common Desktop Environment) was the first promised deliverable of this initiative and after two years its completion was announced at UNIFORUM in March of 1995.

CDE 1.0 was a joint engineering effort between HP, IBM, Sun Microsystems, and Novell that was aimed at producing a de facto and de jure standard in a critical and highly visible technology—the X Windows graphical desktop. CDE 1.0 is about 1.2 million lines of source code and over a thousand pages of end-user, system-administrator, and software-developer documentation. The jointly owned CDE 1.0 source tree also contains over 1200 automated regression tests written in C. These tests were developed to test the CDE code base.

The primary aims of all of the participants involved in creating CDE were:

- To end the many years of battles between the Motif and OpenLook** camps, which helped to fragment the UNIX industry and slowed the development of UNIX applications
- To create a single desktop standard that would provide a common look and feel for users, a common set of administration tools for system administrators, and a common set of desktop APIs for software developers (A list of these APIs is provided in ***Appendix A*** .)
- To lay the foundation for the future evolution of the UNIX desktop in a way that would meet the needs of the open systems market.

The original goal was to take the best technologies from each of the four companies mentioned above with roughly 90% of the code coming from existing products and 10% coming from new work. As it turned out this was not possible, and the percentage is approximately 60% reuse and 40% new code.

This was truly a joint engineering project between four large organizations, which required a whole new software development infrastructure and many new processes. This project was also the collision of four different corporate cultures and standards of practice on project management and software development processes. This article examines some of the challenges that needed to be surmounted, and the processes that needed to be created before CDE could be successful.

History

The odyssey of CDE began in the summer of 1992 when HP and IBM began exploratory discussions on the possibility of creating a standard UNIX environment. In the months that followed they were joined by Sun Microsystems and the UNIX Software Labs (USL), which was later acquired by Novell. The agreement between these companies to work together marked the birth of the Common Operating System Environment (COSE) , and more specifically, the Common Desktop Environment (CDE).

However, on the CDE side, agreeing to work together may have been the easiest part of the process. Deciding what technology fragments would be the starting point, what would be the minimum acceptable functionality, and who was responsible for what pieces took a significant amount of difficult discussion. As we worked through these and other difficult issues it became very obvious that good communication would be what would make CDE successful. To facilitate this communication and to get things going, the following teams (with a representative from each company) were established:

* ONC is Open Network Computing and DCE is Distributed Computing Environment.

** OpenLook is Sun Microsystem's X Window System toolkit.

Management. The management team dealt with contract issues and major allocation of resources (like funding for localization). They were the oversight team and all other committees reported periodically to the management team. Formal project checkpoints were presented to this team and when necessary, they acted as the arbiter for other teams when a deadlock occurred.

Product Description and Requirements. This team was the day-to-day project management team, handling issues such as staffing, technology ownership, and blending the technologies offered from each company in the new UNIX desktop. They were responsible for finding owners for new or unplanned work items and defining the content and scope of the project.

Process. The process team was the heart of the CDE project. It was the job of the process team to determine the software life cycle and defect tracking system to use and the mechanism for tracking and reporting schedule information. They defined the hardware configurations that needed to be exchanged between each company. Their most challenging task was to figure out how to allow access to a single, live source code tree for each partner through their corporate firewalls without violating security constraints. This team was essentially responsible for creating a whole new set of policies that would allow four engineering teams from four different companies to work together productively.

Architecture. While the intent was to use as much existing technology as possible, it was important to determine how that technology would be joined together and then identify what new functionality would be needed to ensure a clean, well-integrated desktop. It was the job of the architecture team to ensure that these things happened and to ensure that individual component owners provided the minimum feature set required by each of the four companies.

Standards. From the beginning it was understood that the CDE project would work closely with X/Open® to submit design documentation and later specifications to help CDE quickly become an industry standard. The standards team was responsible for working with X/Open and other standards bodies to understand their needs and processes and to ensure that we could present our materials in a useful way. This team also acted as a channel for feedback to the other CDE teams.

User Model. Each partner in the CDE project had a significant interest in defining the direction of the user model for the desktop (i.e., its look and feel). IBM had CUA (Common User Access) and each of the other participants had been shipping desktop products with fully defined user model policies. The goal of this team was to resolve disagreements about the general user model, as well as deciding specific questions about component behavior and its level of integration with the system.

As this first set of six teams began to interact and started to deal with the issues, it quickly became evident that we needed more teams to focus on specific issues. Thus, the following teams were added:

Build. This team was responsible for keeping the multiplatform builds healthy at each site and making sure that the code was built successfully on a nightly basis.

Learning Products. This team decided what online help and hardcopy manuals would be written, who would write manuals, what tools would be used, the schedule for manual review, and so on.

Performance. Those of us shipping large UNIX desktops had a wealth of experience which showed the need to be very mindful of performance issues early, when it was still possible to correct performance problems. This team was responsible for establishing benchmarks (i.e., user tasks to be measured), finding bottlenecks in the system using performance tools, and working with the individual component owners to fix problems.

Internationalization. This team was responsible for guiding the development teams to ensure that components were properly written so that they could be localized and that policies were followed so that the CDE code worked on all four platforms and followed industry-standard practices in this area.

Localization. This team managed the process of localizing message catalogs, online help, and hardcopy manuals. They were also responsible for deciding on the subset of target languages, finding translators, getting translations into the source tree, and ensuring that the translations were correct.

Test. This team decided on the test tools to be used, defined and implemented the test scaffold, and created the automated and manual tests that would prove the quality level of the desktop.

Change Control. Halfway through the project, this team was set up to ensure that changes were made to the code in a thoughtful and controlled manner and that the appropriate trade-offs were considered. The change control team was instrumental in ensuring that we were able to meet a variety of delivery dates with stable code. In the final months of the project this team played a critical role in determining what defects would be fixed in CDE 1.0 and what defects and design issues would be postponed until a later release of CDE.

Aside from these teams, many of the component owners set up small intracompany teams to better ensure that they were meeting the needs of the other participants. Examples of this were the teams doing the terminal emulator (DtTerm) and the team doing the desktop services (like drag and drop) who held weekly telephone conferences to ensure that there was acceptance from the other partners on their directions and implementation choices.

All of these teams held weekly phone conferences and exchanged email almost daily—especially the build and process teams. Communication was the most difficult part of this project. Despite the fact that we spent communication resources lavishly, communication breakdowns were the root cause of many of the misunderstandings and functionality problems that arose during the course of the project.

Some teams worked better than others. For example, the process team worked extremely well together throughout much of the project, whereas the test team seemed to have more than its share of problems. Partly, this was caused by the sheer enormity of the test team's task. They were faced with the challenge of creating from scratch a whole test scaffold and infrastructure that would meet the needs of all four participating companies. While API testing is a relatively straightforward process, testing GUI clients is not. Also, the test team was in the unfortunate situation of having to evolve processes as we went along which caused a good deal of rework and other problems. A discussion about CDE testing is provided in ***Article 7***.

Those teams that worked best did so because the individuals involved had the time to come to some level of trust and respect for each other. They learned to seek a common ground and attempted to present a common position to the other committees. A surprising but significant problem for several teams was the amount of employee turnover resulting from reassignments and resignation. This always caused a team to go through a reset, which was very time-consuming.

New Processes

Some of the individuals in each participating company were very devoted to their own internal processes. Thus, in creating new joint processes for CDE we often faced severe internal resistance from the engineering teams at all sites. This was partly because we were all operating under very tight schedules and didn't have the time to cope with new ways of doing things. Another factor was that, not only did the teams have to learn new processes, but they also had to defend these new processes within their organizations.

By the end of the project we did evolve an effective methodology for implementing new processes across all four companies. The first step involved communicating to everyone concerned (engineers, project managers, etc.) the new process and why it was selected. This was done well in advance of when we wanted to implement the process to allow for questions and issues to be raised at all levels and for the process to be tuned if needed. Later the process team personally visited and got acceptance from all of the project managers so that they would cooperate in implementing (and enforcing) the new process. Finally, it usually involved some degree of follow-up from the process team to make sure that the teams were fully engaged in the new process.

A Single Source Tree

The time available to accomplish the project was incredibly short, and the split of technologies was very interconnected so that it was not possible to work separately and make code deliveries to each other. We were all dependent on new functionality being created by the other teams and needed that functionality to fulfill our own objectives. Also, all of the partners wanted to be on an equal footing with regard to access to the source tree. It was conceivable that people could be working in the same areas of the tree so we had to prevent multiple people from working on the same code at the same time.

We agreed that we would put together a single shared source tree that would be live, in that when an engineer checked out a file at Sun Microsystems, it was immediately locked to all of the other partner sites. The major obstacles to implementing this were our corporate firewalls and the bandwidth between our sites. After significant research (and a lot of help from our respective telecom departments) we put in place a frame relay system between all sites. This gave us T1 (1.544 Mbits/s) bandwidths. While it took a long time to acquire the hardware and work the issue through our various corporate security departments, the efforts paid off handsomely. This was a critical key success factor that really enabled joint development between the various companies.

To overcome the firewall problem we created a separate suspect network at Corvallis which had a single machine on it. This machine was the keeper of the golden CDE bits. This machine was set up to only allow UNIX socket connections from four specific machines (one from each site). A daemon written in Perl* ran on this machine and handled RCS (revision control system) requests coming in from each site. At each partner site a series of Perl scripts were written which wrapped the RCS functionality. When an engineer did a checkout, the request was funneled through another firewall machine to the CDE source machine, which did the appropriate checkout, encrypted the file, and sent it to the requesting machine where it was decrypted and deposited in the appropriate directory. Each night the CDE source machine created a tarball of the entire source tree and passed it along to each partner site where it was built.

Each of the participating companies provided approximately six machines to each partner for build and test purposes. Every night each company built on all four platforms. Each company had to develop significant expertise in developing software on multiple platforms.

* Perl (Practical Extraction Report Language) is a UNIX programming language designed to handle system administrator functions.

This simultaneous cross-platform development was not only a necessity, but also very expensive. As engineers made code changes they were required to build and run the new code on all four platforms. Thus, it took a long time to make changes to the code and check it in. The advantage, of course, was that by ensuring that the code always ran on multiple platforms, we didn't find ourselves in a situation where we had to redesign and rewrite functionality at the end of the project to work within different operating system configurations.

Decision Making

It was clear from the beginning of the project that we would not get anywhere if we used a consensus model for decision making. Therefore, we decided to use essentially the model created by the X Consortium. In this model a representative from a single company became the architect for a technology (e.g., Motif toolkit) and was responsible for designing, documenting, and creating a sample implementation of a component or library. Although the architect was required to be open to input for requirements and needs from the other participants, the final say regarding that technology was the responsibility of the architect.

This was the model we agreed to use and in large part we were successful at using it. However, there were times when we ran afoul of this process and the technical decisions of component owners were challenged by one or more of the other participants. These challenges often were escalated all the way up to the management team where, because they were distanced from the technical problem at hand, it took a significant amount of time to work through the issues. It was recognized by all parties that future efforts would make better progress by defining a single point of authority for making final, binding rulings on technical issues.

Scheduling

When the COSE initiative was announced, the schedule for the CDE project was also announced. This schedule was intended to be very aggressive and also reflected the initial assumption that 90% of CDE would be existing code with only 10% new code. The reality was that the melding of the technologies from each of the participants required extensive rewrites in many areas. Also, the functionality was somewhat fixed since none of the participating companies felt they could end up with a desktop that was less functional than the ones they were currently shipping.

Starting with the desired end point, the process team created a schedule that had alpha, beta, and code complete milestones followed by a final test cycle. We needed to define exactly what each of us meant by these milestones in terms of functionality, defect status, localization status, and branch flow coverage. We created checklists for component owners to fill out and attempted to normalize the data we received from individual project managers.

These first scheduling efforts were flawed for a number of reasons. The first was that coordinating the efforts of four different companies was very time-consuming. Second, we were unprepared for the scope of new processes we would have to put into place and the amount of time it would take to implement them. In some cases we made the classic software management error of trying to legislate the schedule, functionality, and effort with predictable bad results.

We eventually understood the kind of overhead we were dealing with and were able to create a viable schedule that was used to good effect.

Conclusion

There were hundreds of aspects of the CDE project that could have been discussed. Only those issues which in retrospect seemed most important have been discussed. Participating in a multicompany joint development project is a very challenging affair. It requires a willingness to rethink all of the familiar software development processes and tools and to come up with new solutions from both management and engineering teams. It requires an open mind in listening to what each participant is saying since they may use the same terms but with different contexts and meanings. The importance of communication at all levels cannot be stressed enough. Taking the time to build personal relationships at all levels will pay back dividends over the life of the project.

In the best of circumstances, the work being done by the other partners must be constantly monitored to minimize misunderstandings, ensure that commitments are being met, and help shape the inevitable trade-off decisions that occur during a project. Also, it's important to plan for the fact that disagreements and escalations will happen in the most amicable of projects. While it means some loss of control, creating a neutral single point of authority for resolving these escalations will save enormous amounts of time and help to preserve the necessary good will between the participants.

It's natural to underestimate the time it will take to put new processes in place, to develop software on multiple platforms, and to communicate and work out issues with joint development participants. This tendency to underestimate joint project overhead will also appear in individual engineer's schedule estimates. Of course it is absolutely necessary that the engineering teams have the opportunity to resolve ownership and functionality and do a bottom-up schedule that can then be discussed in terms of trade-offs of functionality, performance, and other features before the project schedule is decided upon.

In most cases the work will be done by engineering teams from a variety of geographic locations. Usually these teams do not know each other—except as the competition. Yet they will need to work productively together to make each

other successful. It will almost always be worth the time and money to bring the teams together face to face to build a relationship that will prevail through the inevitable tensions of a joint development project.

Finally, it's important to realize that, despite the problems, joint development can be done. Further, because of the diversity of experience and abilities of the different participants, the end result will be richer and appeal to a broader set of customers.

▶ Go to Article 7

▶ Go to Table of Contents

▶ Go to HP Journal Home Page