Depo. Ex. 1109

TO: BradC, BradH, BradSi, JohnEn, PhilBa
FROM: MikeDr
DATE: 9/18/91

Attached is a copy of the DOS 5.0 Post-Mortem Report for your review. I consider this to be in approximately its final form, but as you were important participants in the DOS 5.0 development process, I want to give you an opportunity to review and comment on this before I release it generally. It has already gone through a review by the participants in the post-mortem meeting, and the feedback I received has been incorportated.

If you have any comments or corrections, please let me know by next Wednesday, 25 Sept., so that I can incorporate them into the final version for release next week.

Thanks.

MS-DOS 5.0 Development Post-Mortem Report

Prepared by Mike Dryfoos

Executive Summary -- Summary of Recommendations

Members of the DOS 5.0 team conducted a post-mortem examination of the DOS 5.0 project in July 1991. This post-mortem meeting focused on the development process, but included representatives from the other relevant functional groups, and examined a wide range of project process issues.

The post-mortem discussion was organized around three key questions: what worked well over the course of the DOS 5.0 project; what didn't work well; and what should be done differently next time. The main body of this document is organized in the same way. This summary of recommendations lists the key points arising from the discussion as a whole, on how we can better organize and direct our efforts to make the next project run more smoothly.

Product Definition and Specification

Define clear objectives

The DOS 5.0 project maintained clear and consistent objectives from its inception in December 1989 until its conclusion. These helped focus the team's efforts and sustain morale. All projects should begin with clearly defined goals.

Get adequately broad input into product definition

The original DOS 5.0 specification was largely the creation of Development and Program Management, working in isolation. Other groups' needs and desires were not adequately addressed at the beginning. Future projects should ensure that all relevant constituencies are adequately represented in the product definition phase.

Spend adequate time on feature set definition

As a corollary, adequate time should be devoted to defining the product feature set, to allow adequate input, and to consider the implications of design and feature choices. The DOS 5.0 project team rushed through its specification phase, resulting in additional work later when it was realized the spec was incomplete.

Include doc, testing, and localization plans in spec

To help ensure completeness of the specification, and to provide a clear view of the entire project schedule, all constituent teams should include their product plans in the main specification, rather than have the spec be centered solely on development.

Consider both retail and OEM sides of product

MS-DOS 5.0 is both an OEM and a retail product, but the project team focused much more heavily on the retail side. More attention needs to be paid to the OEM deliverable, the OAK, to ensure that it adequately meets the OEM needs.

Keep specs up to date and on line under source control

As with most projects, keeping the spec up to date proved difficult over the course of the development phase. Time needs to be specifically allocated for spec updates. The specification document should be part of the project source tree.

Schedule periodic specification reviews

Periodic specification reviews should be built into the project schedule, to ensure that the product content satisfies the product goals over the course of the project.

Scheduling

Include all project components in schedules

During DOS 5.0 development, the time required for some project components, such as documentation review, were not adequately reflected in the working schedules. All project components that affect development and testing should be included in the overall schedule from the beginning, to avoid surprises later.

Conduct periodic schedule reviews

As with the specification, the schedule should be reviewed on a regular basis, to ensure its accuracy and completeness.

Exploit historical data to create base line estimates for bug open and resolution rates

Where possible, we should develop metrics based on our experience for use as predictive tools in the future. The bug open and resolution rates from DOS 5.0 can be extracted from our bug database and analyzed for this purpose.

Communications

Maintain small teams to help foster flow of ideas and information

An important contributor to the success of the DOS 5.0 project was the free-flowing communications fostered by the small team size. Even with larger projects, we should strive to build small team identification and informal contacts for greatest productivity.

Hold occasional meetings of entire group to update status

Occasional meetings to update the entire group on progress and plans can help foster a shared sense of purpose. Project kickoff meetings can be helpful to reinforce overall goals and clarify responsibilities.

Publish lists of experts to speed up contacts among groups

Especially in larger groups, accessible lists of experts in specific areas can help get questions directed to the people who can answer them as quickly as possible.

Schedule seminars to build up general expertise

We all need to continue our education about our product, its competition, and the way it is used by our customers. Seminars featuring members of the project team presenting topics they know well will help build up the overall expertise of the group.

Forum (*née* War Team)

Forum should be constituted at project start

In the DOS 5.0 project, the war team or forum was introduced only in the later stages of the project. In the future, this management group should operate from the beginning, so that there is always a clearly designated body for resolving problems.

One representative from each functional group

The initial membership of the DOS 5.0 war team did not include representatives from all of the component project teams. Representatives from excluded teams were added later, when it was realized that their participation was required. In future projects, a representative from each component group should be included from the outset. Participation should be restricted to one individual from each functional group, to keep the forum down to a manageable size.

Agenda and minutes distributed regularly and broadly
Forum meetings should be preceded by publication of an agenda via email to the entire project team. Minutes from forum meetings should likewise be distributed promptly and widely.

Beta Program

Large scale beta distribution
The large beta program was one of the chief successes of DOS 5.0. We should continue to plan similar efforts for future systems products.

Compuserve support forum
The Compuserve support forum was an excellent channel of communication between the beta users and the project team. A similar bulletin board should be used for future beta support.

Collecting data from beta sites
Configuration information collected from beta sites proved valuable in unexpected ways, for resolving issues and bugs. We should make sure we continue to include ways to get this kind of information from our beta sites. Automatic collection and delivery methods for the information of interest should be designed into the beta software.

Expand international beta test
Greater international participation will help us identify problems with our international support. Future international beta tests should be initiated earlier in the project, and distributed more widely.

Test and Build Process

Testing resources should be augmented
The testing team needs more resources to adequately test the range of conditions DOS is expected to operate in. More hardware will extend the configurations we can support; more personnel will help us reduce our reliance on temps and external test houses, both of which had mixed performance in the DOS 5.0 project.

Testing should work to improve their efficiency
The testing team should continue its efforts to automated its work, in order to extract best use of the resources it has. Automated kernel regression tests should be created and made available to developers for quick sanity checks.

Regular builds and smoke tests should be continued
Regular builds and smoke tests were valuable for providing structure for development and identifying problems quickly. They should continue to be part of our routine development cycle.

Developers should incorporate white box testing
Developers should enhance their use of active testing and debugging techniques while writing code, in order to identify problems before software gets into the hands of testers.

Build function should report to Development
The build function should be reassigned from Testing to Development, in order to reflect the product orientation of the builder's work. Remember that the builder's output is the product, as far as the OEMs are concerned.

Lab maintenance should be improved

Some time should be allocated by the testing team to keep the labs in reasonable shape. Some degree of control over machine reconfiguration needs to be instituted, to keep our hardware resources in good shape.

Documentation

Devise better technical review process

The documentation review process didn't work well over the course of the DOS 5.0 project. Reviews were not adequately accounted for in the schedule, and the quality of the reviews was low. Development, Program Management, and User Ed need to devise a better review process for future projects.

Integrate programming writers into project team

The quality of the DOS 5.0 technical documentation suffered because the programming writers were not well integrated into the project team. In future projects, they should be explicitly assigned to the project, and their work should be part of the project schedule.

Transfer ownership of OAK guide to User Ed

The documentation accompanying the OAK has suffered from lack of a specific owner, and that shipped with DOS 5.0 is of poor quality. The programming writers on the User Ed team need to take responsibility for making this a useful and usable part of the product deliverables.

Compile PSS notes in advance of project release

User Ed can help PSS prepare for anticipated problems by writing PSS technical notes during the project, as we become aware of issues that will not be resolved by the code. Early preparation of these materials will help reduce the sort of support load we experienced at the release of DOS 5.0.

International

Develop international expertise in domestic teams

During DOS 5.0, the domestic development, program management, and testing teams largely relied on IPG to identify issues and provide solutions to problems faced by our international customers. However, IPG wasn't in a position to play that role. The domestic teams need to develop specific expertise in the international arena, and get closer to the international customers.

Regularize schemes for handling localizable text in source code

The DOS 5.0 source tree is not organized in a way that facilitates easy localization. Too many different schemes for handling localizable messages are in use. This needs to be regularized for the next version.

IPG should try to anticipate problems, intervene early

IPG should strive to take a more activist role, and intervene early when it can see problems arising from the way the product is being developed.

External Dependencies

Establish and apply clear quality guidelines for external acquisitions

External acquisitions included in DOS 5.0 suffered from poor to mediocre code quality, and required extensive modification for localization. Specific guidelines for acceptance of external

acquisitions need to be established, with special focus on code, internationalization, and usability, and usability reviews included prior to acceptance.

Establish clear communications with other MS groups via Program Management

Where we have dependence on other groups inside Microsoft, Development and Program Management need to communicate our needs clearly, and ensure that we are receiving adequate priority and attention from the other groups.

Tools

Improve the bug tracking system

Performance and reliability problems with the bug tracking database used during DOS 5.0 caused some noticeable losses of productivity. Before beginning bug tracking for the next project, we should make sure the database tool offers sufficient strength and capacity to do the job.

Build database of source code routines

Increasing our use of common source code libraries, and building up a database of common routines, can help improve our future productivity and reduce bugs, especially with regard to the DOS utilities.

Introduction

This is a report on the outcome of the DOS 5.0 post-mortem meeting conducted on 19 July 1991. The purpose of the post-mortem was to review the process used to develop the DOS 5.0 product, to identify where the process worked and where it didn't, and to suggest improvements for the future. The meeting was oriented towards the development process, but other functional groups were represented.

This document is based on the comments and notes made at the post-mortem meeting along with other feedback received before and after the meeting. It is not strictly an account of the meeting, but rather commentary on and explanations of the issues raised there, with some general and specific recommendations. While I have tried to present the full range of opinions expressed by the post-mortem participants, naturally some personal bias is to be expected. The actual notes from the meeting are attached as an appendix.

The participants and their team affiliation were as follows:

Development
Mike Dryfoos
David Olsson
Harish Naidu
John Hensley
Sriram Rajagopalam
S. Harikrishnan
Scott Quinn
Chuck Strouss
S. Mohanraj
Nagarajan Subramaniyan
Dave Berliner

Testing
Jim Landowski
Terri Bronson

Program Management
Tom Lennon
Eric Straub
Fernando Garcia-Duarte

User Education
Marion Junttila
Scott McMahon

IPG
Dave Stanchi
Steve Blanding

Steven Timm, facilitator
Greg Tzinberg, record keeper

The meeting focused on the three questions, as viewed by the various functional groups represented:

- What aspects of the process worked well?
- What aspects didn't work well?
- What should be done differently in the next project?

This document is organized around these same questions. Discussions of the good and bad aspects of project organization and components, and recommendations for future improvements, are all discussed separately.

A Brief History of MS-DOS 5.0

Prehistory

The DOS 5.0 project came together as the combination of two distinct but related goals: to provide for an upgrade to the current version of DOS, to be sold through the retail channel; and to provide an attractive replacement for DOS 4.01, eliminating the quality problems and the poor user and industry perception of the latter.

Of the two, the upgrade idea was the older. Its genesis lies in a memo from Tom Lennon to Steve Ballmer in September 1988, discussing an upgrade product based on DOS 4.0. Among other highlights, this preliminary proposal called for a team of two developers and one tester, and development time of 40 manweeks. Subsequently, work began on the upgrade installation program and improvements to the DOS 4.0 shell for this project.

Eventually, it became apparent that offering DOS 4.0 in the retail upgrade package would not work since the market acceptance of the OEM DOS 4.0 product was so poor. We saw the importance of "adding sex" to the product and overcoming the liabilities of DOS 4.0. The original plan was modified to include improving the utility set and rewriting the DOS shell.

Meanwhile, IBM was making plans to continue developing new DOS versions based on DOS 4.0, to meet their new hardware needs and to track changes in OS/2. A version called "Lifeboat", including features such as extended attribute support and an improved shell, was developed, but never came to fruition. It was replaced by two other plans, "Captain" and "Jetski". Captain was to be a merger of DOS and Windows into a new, more tightly coupled system, complementary to OS/2. Jetski was an improved version of DOS, making use of the High Memory Area to reduce the DOS memory footprint, and incorporating numerous bug fixes to DOS 4.0.

Captain eventually faded away, falling victim to lack of interest and the deteriorating MS-IBM relationship. Jetski persisted longer, and our development team spent considerable time exchanging and merging code with IBM, but conflicts over the HMA implementation, IBM's rejection of our new utilities and shell, and lack of convergence with our retail upgrade plans, created difficulties. We were left without adequate confidence that we were developing the right product. We saw the need to move aggressively to improve DOS, in particular by reducing its memory footprint, but we felt we would not be able to do so while trying to collaborate with IBM.

Finally, SteveB and BillG succeeded in persuading IBM to turn over all DOS development responsibility to Microsoft, and in December 1989 we were granted permission to proceed with our own specification for what was now called DOS 5.0.

Early Stages

The initial feature set of DOS 5.0 combined the new utilities and installation program from the upgrade effort with plans to move the DOS kernel to the HMA and otherwise reduce its size. The development effort began with a brief design phase in which the features and their implementations were devised, and schedules created. The development team was responsible for creating this preliminary specification and schedule, which was then massaged by Program Management into its final form. The development spec was largely complete by the end of December. The schedule projected coding would be complete by the end of March and the product ready for final release in early August.

At this time the key features of DOS 5.0 included the following:

- DOS in HMA on 286 and 386 machines
- Removal of DOS 4.0 kernel features
- New shell
- New upgrade and OEM installation programs
- Enhanced utilities: DIR, Safe FORMAT, DOSKEY
- ROM executable kernel
- VCPI and VDS compliant EMM386

Work proceeded well, and for the most part we made our original code complete date, despite losing one member of the team on an extended medical leave, and another to a special OEM project, the Flash File System. However, the feature list had already started to lengthen, as new items were added to the product, and unanticipated problems had to be addressed.

Changing Product -- Changing Schedule

The additions to the DOS 5.0 feature set came from a variety of sources and circumstances. Some were opportunistic additions that appeared to cost the DOS team little (QBasic). Some were optional features in the original spec that we decided to include (online help). Some solved problems we hadn't thought through in the original design (Setver), or reflected a change in product orientation (the DOSShell interface). One of the most important stimulants for adding features was competitive pressure from DRDOS 5.0, which we first learned of in the spring of 1990. The DRDOS feature set led us to add UMB support, task swapping, and Undelete.

Unfortunately, it took us some time to revise our schedules to match the changing product. We adjusted the schedule outward in small increments, and the end dates lost clarity and credibility inside and outside the team. Considerable amounts of the team management's attention was diverted to new features such as file transfer software, undelete, and network installation, dissipating our focus on getting the product done.

Eventually this situation reached a crisis point at the end of July 1990, and, led by BradSi, the team's management spent an arduous series of meetings nailing down a schedule and process for closing the project down. A third and more extensive beta release was added to the schedule. Release projections were made on the basis of the expected incoming bug rate, the size of the bug backlog, and the projected bug fix rate. The development team was brought into the process for their input and buy-in to the newly revised schedule, which called for the final beta release in October, and RTM in December 1990.

Final Stages

The final stages of the project are remembered largely as a long grind through the bug list coupled with struggles to get beta releases stabilized. We had one notable disaster when we released an internal beta with a Setup bug that corrupted users' disks in some configurations. Subsequently we became more cautious with regard to Setup changes. The bug backlog was reduced more slowly than expected, due to higher than anticipated beta bug rates and difficulties working with beta sites to get adequate information to reproduce their problems. Over time, team management exercised increasing control over which bugs were eligible for fixing and developers spent more time reviewing changes before they were incorporated. Regular bug list cutting sessions became a feature of the project activity, and eventually all bugs required management approval to get fixed.

The final beta release, sent out to thousands of sites, was about two months late. We released it about the time we had hoped in July to ship the final version. Its positive reception, however, helped build the team's confidence in the quality of the product, and as we moved forward from that release, we knew we were getting close to shipping. The announcement date was set for 11 June 1991, and the RTM for early April. The last bug approved for fixing, a minor change to the FORMAT program, went in on 1 April, and RTM finally happened on 12 April. Just before RTM, DOS 5.0 became the first product ever in the DOS/Windows group to complete its full planned final system test successfully.

What Worked Well?

The project history provides background for understanding the issues raised during the post-mortem discussion, which was concerned with process, not events. Our starting point was a review of the aspects of the project that worked well or contributed to its success.

Focus

Despite a drifting specification, the project had clear, consistent objectives from its inception in December 1989 to its final release. In particular, the continual focus on product quality, DOS compatibility, small memory footprint, and improved ease of use provided a clear theme that the team could rally around. These goals provided identifiable metrics that allowed us to check our progress, and helped build support for processes that reinforced the objectives, such as ongoing code reviews. They also helped support the team's morale, even when the project seemed to be dragging on and on, because we knew that we were going to deliver a quality product.

Cohesiveness and Communication

Clear objectives fostered a common sense of purpose, which in turn built up the cohesiveness of the team. Individual developers avoided narrowly focusing on their assignments, but instead participated widely in exchanging ideas on design and implementation. Typically, individual assignments were varied and changing; for the most part, team members adapted graciously to shifting circumstances. The development team maintained a sense of shared global responsibility for the product content and quality, rather than each individual paying attention only to his own small piece. Shared responsibility led to a shared sense of pride as we came to understand what we had accomplished, and how it was received by our beta community.

The relatively small size of the development team helped foster our internal community. Everyone knew everyone else, and knew who could answer questions in any specific area. We were able to work together well with a minimum of formality, and to exchange ideas freely.

This sense of openness extended beyond the development team. The relationship between Development and Testing was cordial and cooperative. The generalist nature of the development team was reflected in the testing team. Arrangements where individual testers and developers worked together to regress specific problems worked well.

User Ed and IPG also had their recognized role on the team. UE was given responsibility for nearly all the text seen by the outside world, not just the printed documentation; this brought them into closer contact with the development team, and helped build relationships. UE was encouraged to contribute their viewpoint, and was able to represent the average user's perspective. Documentation was used as a tool to resolve some product problems, helping to relieve the development team of some of its burdens.

While some confusion existed about IPG's role, the IPG team and the developers were able to work well together to resolve problems in the code. Although the organization of the source tree created problems, most of the code that had to be internationalized was easy to work with, and developers were prompt and responsive in addressing concerns raised by IPG. IPG did a good job seeking out the information it needed, rather than waiting passively for it to be delivered. As the project progressed, IPG came to be viewed as part of the team, rather than an external client.

Build and Test Process

Several aspects of the build and test process helped provide structure for and quick feedback on development efforts. Regular weekly builds defined targets and milestones for developers as they worked to get particular features into the product. Smoke tests immediately following builds identified bugs quickly, increasing the testing team's productivity. Stress tests involving all of the DOS group or all of DOS/Windows helped identify problems with our network installation scheme early on, avoiding a company-wide distribution before we were really ready. Likewise, our policy of targeting groups within the company as preliminary internal beta sites helped iron out problems before we went to wider distribution.

The test team understood the implications of releasing MS-DOS as a retail product, and responded with a testing effort that surpassed that on all previous DOS versions. Special focus was paid to compatibility, with over 300 applications being tested in multiple configurations. Testing of Setup was exhaustive, with over 100 machine configurations checked. The test team's efforts paid off with a very high quality final release.

Testing increased its use of automated testing over the course of the project, helping to reduce the team's workload and improve their efficiency. The development of the DTS engine and utility test scripts for it improved the smoke test process and freed resources for other tasks.

War Team

The war team was regarded as a mixed blessing by the team, and will also be discussed in the section on what didn't work well. It was generally agreed that effectiveness and contributions of the war team improved as the project went along.

The war team was established by BradSi's directive, shortly after he started. It included representatives from all of the team constituencies. The war team helped provide focus and leadership for decision making and followthrough. It provided a forum where team members from all constituencies could give input on important product issues. Distribution of the notes from the war team meetings helped keep the full team informed of decisions reached and directions established.

Although not strictly part of the war team activities, certain jobs taken on by subsets of the team helped keep the project on track. The regular bug purging sessions helped reduce distractions and eliminate tasks of lesser importance. The documentation walkthroughs helped questions concerning the manuals get answered quickly and accurately.

Beta Program

There was general agreement that the beta program was one of the best aspects of the DOS 5.0 project. The beta team has had its own post-mortem, and it won't be discussed in detail here, beyond hitting some of the high points on how the beta activities affected the rest of the team.

The beta program provided great opportunities for direct contact between the development team and the customer. The Compuserve forum was an excellent source of input. The Uninstall disks returned by the beta testers provided a valuable database for investigating several problems. The large scale of the beta test helped turned up many problems that would have been missed had we relied on our own testing resources, and helped ISVs and IHVs get ready to support DOS 5.0 as soon as it shipped. The beta program also helped build excitement and word-of-mouth interest in the product prior to shipment. The administration of the program was smooth and reliable.

Summary of What Worked Well

- Clear and consistent focus, with measurable goals
- Open communications within and among teams
- Small, cohesive teams
- Structured, predictable build and smoke test process
- Extensive testing effort
- Clearly identified leadership and decision-making body
- Well run and extensive beta program

What Didn't Work Well?

It was easier for the post-mortem participants to identify things that didn't work well than those that did, and consequently this section is longer than the preceding one. Some of the most heated discussion centered on scheduling, but a number of important issues were identified, and are discussed in this section.

Inadequate Specifications

Much of what ultimately went into the DOS 5.0 project was not included in the initial specification, or indeed ever spec'ed at all. Keeping the spec up to date with a changing product is a problem for almost every project, and adding features late in the game is a common occurrence -- indeed, our ability to adjust product goals and features midstream can be a strength. However, it is worth noting that the most successful part of the DOS 5.0 project, in terms of adherence to defined schedules, was the first three months, when we were implementing the features spelled out in the original specification. We got into schedule trouble later on, as we continued to modify the product's feature set without updating and reissuing the specification.

In part this reflects a lack of discipline on the part of the team leads, their failure to keep the project team focused on the original goals. More significantly, it reflects the inadequacy of the feature set described in the original specification. That spec was principally the creation of Program Management and Development, with some input from Testing. Product Management, User Education, and IPG were largely left out of the original product definition process. They were left to catch up later, at which point hearing their concerns and addressing their needs meant unexpected development and testing time and effort.

Adding features to a product already spec'ed is not necessarily a bad thing, and few of these additions to DOS 5.0 could be considered gratuitous. However, we often failed to consider the full impact of any particular change on the project as a whole. Some changes were spec'ed very informally, some not at all. Affected groups, especially Testing and User Ed, were not kept informed, and so were surprised by additional work. Features were added without adequate consideration of their value relative to their schedule impact.

Another characteristic of late or ad hoc changes is that they tend not to be thought through or reviewed as carefully as those originally spec'ed. On occasion, we made choices that later had to be reversed or modified, because we hadn't considered all the consequences. For example, when we added support to detect top of memory in some of our device drivers, we didn't examine the impact on third-party device-driver loaders. As a result, we had a panic very late in the project when we realized there were problems.

Schedules

Late feature additions contributed significantly to schedule problems, as noted above. Lack of coordination among teams and failure to consider the full ramifications of changes created many difficulties. In addition, we suffered from confusion about schedule ownership, personal and mechanical difficulties with schedule building, and inadequate scheduling of non-development activities.

Although the team leadership tried to be consistent and clear in its position that Development created work estimates that Program Management assembled into a schedule, there persisted among the development team confusion about how the schedule dates were arrived at, and suspicion that the team's input was not adequately considered. As a result, the developers didn't always feel the sense of ownership of schedule commitments that management, especially Program Management, expected.

Individual milestones would go by with frustration on one side that goals had not been met, and on the other that expectations had not been realistic.

Estimating the time required to accomplish any particular task is a skill usually acquired through experience, and with the mostly young DOS 5.0 team, it isn't too surprising that some time estimates were incorrect. However, it did seem at times that individuals were confused about how Program Management intended to use their time estimates. Some individuals produced estimates that represented best-case scenarios, rather than realistic ones, and then were surprised to see their best-case guesses show up on schedule charts. Others felt a lack of trust when they found their estimates questioned by Program Management. Better explanation of the goals and methods of scheduling could have helped clear up some of these problems.

Scheduling difficulties weren't confined to the development team. Sometimes other groups' needs were not adequately accounted for ahead of time, leading to delays when they finally were addressed. The best examples were documentation reviews; these were time consuming and demanding, but sometimes came with inadequate warning, and were usually not reflected in the overall schedule.

Interim Milestones Too Vague

Another aspect of drifting specs and unsteady schedules was uncertain milestones. Significant dates on the schedule charts, such as code complete and beta shipments, weren't always crisply defined. We didn't always know when something would be done, or pieces would be completed at different times, making the achievement of the specific milestone a vague, poorly defined event. At other times, a well-defined milestone would necessarily be postponed, but without sufficient deliberation on the new date. The many weeks required to ship the third beta release are the best example of this -- problems kept arising, the date kept slipping, but we all continued to act as if the shipment was just around the corner.

When interim goals were achieved, they tended to pass with little note. We didn't exploit these opportunities to celebrate, commemorate, or recoup, but instead just went back to work with the feeling that we were farther behind than ever. A complaint was raised in the post-mortem discussion that we didn't have enough fun over the course of the project; the interim milestones provided occasions that could have been used to boost morale and unify the team.

Pep Talks and Negative Feedback

Attempts to motivate the team via "pep talks" were generally unrewarding. The talks didn't succeed in conveying a sense of excitement or purpose about the project, in the way that direct feedback from customers did. Efforts to rouse or challenge the development team with negative feedback also largely failed; the criticism offered left the team feeling unsupported and unappreciated.

War Team's Lack of Openness

As noted above, the war team did come to be viewed as an asset to the project over time, but it also had its problems, especially in the first few months of its activities. In particular, the project team felt that the charter of the war team was mysterious, its discussions closed, and its decisions unexplained. The war team was intended to provide a forum and a focus for broad participation in decision making, but this was not adequately communicated, and so the team came to view it as a dictatorial body making decisions in a way that the project team members couldn't control. The war team became much better accepted once its minutes and agenda were distributed widely and regularly.

Internally, the war team suffered from an inability to resolve issues firmly. Discussions would recur, decisions be reexamined, leading to considerable frustration and loss of productivity. The painful and extended examination of the network software upgrade question is the most conspicuous example.

For a time, there were parallel meetings of the war team and a somewhat larger project leadership group, which pre-dated the war team. This was another productivity drain, as issues tended to get covered in both forums. Eventually, the war team took over most decision making, and the larger status meetings degenerated into reviews of testing and support status. Eventually, the status meetings were stopped altogether. Arguably, this should have been done earlier, saving everyone's time. On the other hand, it wasn't possible to cancel the status meetings until all the constituencies represented there were also represented on the war team. The initial narrow composition of the war team was another symptom of our failure to include all the affected groups in our decision-making process.

Testing

This post-mortem meeting was focused on the development side of the project, so testing issues did not receive a comprehensive examination. However, there were several important issues that were discussed.

The most important problems connected to testing were in relation to resources. The test group lacked adequate resources, both personnel and equipment, to address the task they had thoroughly. DOS is universal: it is expected to run on every OEM machine, foreign and domestic; on every no-name clone; with every piece of add-in hardware; with every software package ever written. Of course, verifying this would be an impossible mandate for any test group, but we felt (and continue to feel, considering our test constraints on our current projects) that our test group needs more resources at its disposal to do its job well.

Several examples demonstrate the problem: the bugs relating to video adapter detection in Setup, which slipped by because we lacked an adequate range of different hardware; problems with third-party partitioning schemes, which didn't receive adequate testing until very late in the project cycle; bugs in the task swapper that were not revealed because real-life scenarios were not exercised sufficiently.

In some cases, such as the task swapper example, we had hoped that external testing resources would help us identify bugs. As it turned out, the external testing house identified few problems for us, and even when they did, they didn't always provide enough information to reproduce them here. The external testers didn't produce many results for the price, and succeeded mostly in giving us a false sense of security, as the task swapper problems demonstrated. They did not prove to be an adequate substitute for in-house efforts.

The testing group can take some steps to improve its efficiency and effectiveness that don't require resource augmentation. The possibilities for automated testing can be exploited more fully; a perhaps excessive emphasis on bug regression left too little time for more creative testing techniques to be applied.

A more minor but still annoying problem through the course of the project was the continual state of disrepair and chaos in the test labs. Machines were left in pieces, equipment was non-functional, old papers and disks were strewn around. Both Development and Testing share responsibility for the condition of the labs, but Testing is explicitly charged with keeping them in good repair. The comparatively pristine state of the Windows test labs shows that a heavily used lab can be kept in good order, even with a much larger number of people involved.

Documentation and Documentation Reviews

As with the testing discussion, the consideration of documentation in the post-mortem was mostly restricted to its relation to Development.

The documentation teams had their own histories of shifting goals over the course of the project, and even wholesale changes in personnel. The initial goal for user documentation was to produce a book of only modest quality, with minimal resources. The contractors used in this phase of the project left something to be desired in terms of the quality of their work; in particular, they didn't seem to be adequately familiar with DOS itself. The development team wasn't kept apprised of their activities; when we started receiving calls from people we'd never heard, asking detailed DOS 5.0 questions, we weren't sure how to respond. The time we had to spend with them was totally unexpected and unscheduled, as well.

Later, when the writing was brought back in house, things proceeded more smoothly, at least until it came time for documentation reviews. As noted previously, technical reviews sometimes arrived without sufficient warning or accommodation in the project schedule; the documents also tended to be of intimidating size, discouraging a thorough review in the allotted time frame.

On the other side, the documentation team generally found the resulting reviews to be of poor quality. Comments from the team were contradictory or incomplete, reflecting varying amounts of attention paid to the review task, and varying competence of the reviewers to analyze the material they were given. The unevenness of the reviews necessitated a further set of documentation walkthroughs with small groups, to clear up the confusion.

Although relations between the user documentation team and the rest of the project were sometimes problematic, they were much better than the relations between the programmer documentation team and the rest of the project. The programming docs were started very late, there was a wholesale change in the writing team personnel, there was little communication between the writing team and Program Management and Development, and technical review time was inadequate. Partly these problems can be traced to the confusion surrounding the OS/2 reorganization, but it does appear that there was a fairly serious breakdown in communications here. Although ultimately the writing team did a good job on their assignment, there were a number of errors and organizational problems in the book that could have been avoided had there been more time for review.

Finally, as always, the OAK documentation suffered from lack of ownership. The OAK doc was and still remains seriously out of date, and is only modestly helpful to OEMs who want to modify MS-DOS. No particular team has assumed responsibility for the OAK doc, and so its state isn't really a surprise. Fortunately, this is receiving some attention in the DOS 5.00 ROM addendum, through a collaborative effort of Program Management, Testing, and Development, but a thorough overhaul is required.

Internationalization

Examination of the relationship between IPG and the rest of the project revealed three significant problem areas. The first two are smaller scale "project" issues, while the third is a larger "relationship" issue.

First, IPG suffered particularly from the communication problems discussed earlier. Effort expended on translating specs and docs was simply thrown away when the documentation strategy changed. Had IPG been more closely connected to the rest of the project, this waste could perhaps have been avoided.

Second, the source tree was not organized in a way that made localization easy. Four different messaging systems were in use for different pieces of the product, complicating the translators' job. Translatable text, while isolated for each separate program, was scattered through the source tree, rather than centralized, necessitating a tedious exercise in identifying each file containing translatable text. The development team should have been more aware of how the source organization affected IPG, and made it more of a priority to clean things up.

The relationship issue was of broader scope. Fundamentally, the domestic project team and IPG didn't understand how each viewed the other's role. Unvoiced assumptions led to a failure to consider the international aspects of the product as thoroughly as they deserved.

IPG considered itself to be a translation shop, and looked to the domestic team to understand and address international product concerns. The domestic team viewed IPG as the voice of the international constituency, and looked to them to identify product problems in the international setting, and propose solutions. This confusion affected Development and Program Management, with regard to product features, and Testing, with regard to testing of international features, hardware configurations, and applications. As a result, significant international problems, such as the case-mapping confusion, went unaddressed in our product planning, and by the time IPG saw the need to raise our consciousness on these issues, it was either too late to address them, or possible only through a last minute panic.

The international beta test was an important part of the test cycle, and helped reassure us that the original product goals were met in the international as well as the domestic setting. However, the beta would have been more helpful if we had included international sites sooner; this might have called our attention to the international product issues before it was too late to address them properly.

Build process

The build process was a source of a few problems over the course of the project. Network errors and failures to identify build requirements sometimes left us in a state of confusion. Principally, the difficulties seemed to be organizational; the build function fell in between Testing and Development, and there was some uncertainty about who had responsibility for ensuring that builds went smoothly. There was also a lack of adequate supervision of the builder himself; when he had questions or problems, it wasn't clear who should address them.

Related to the confusion over build ownership was confusion over the role of the OAK. In the OEM product, the OAK *is* the product; however, in focusing so intensely on the retail package, the team neglected the OAK. It didn't receive the attention it deserved as a key deliverable .

Outside Dependencies and Acquisitions

We relied on other development groups elsewhere in the company for QBasic, Edit, and CW, and we contracted with parties external to Microsoft for the CPS utilities, Systemsoft ROM DOS features, and other items. Their uneven quality suggest that we need to be more rigorous in our review and control of external acquisitions.

While our relationship with the Basic group was smooth, we did experience many problems with CW in terms of quality and delivery. Their support for our efforts varied, and we generally had to consume a lot of energy to get what we needed. This is probably due to an absence of CW Program Management, and our low position in their priorities.

The external acquisitions reveal more systematic problems. The code we received from CPS, even though it was part of their shipped product, proved to be of mediocre quality and contained a number of bugs, most of which remain in DOS 5.0. Most problematic for us was the difficulty of localizing this code. It had not been written with localization in mind and considerable effort was expended both by IPG and Development to get around this.

The Systemsoft deliverables suffered from flat-out poor implementation. Unfortunately, we did no significant testing until long after we received them, and so didn't discover the problems until fixing them became a panic situation.

Tools

While we had few problems with most of our development tools, the bug tracking database left much to be desired. We suffered from its poor performance, instability, and lack of flexibility. On more than one occasion, a database crash meant the loss of hours or even days of work. In addition, relying on a single administrator for both DOS and Windows meant that updates were not always timely for one group or another.

Summary of What Didn't Work Well

- Incomplete and unstable specifications
- Communication of product changes among affected groups
- Confusion about scheduling goals and methods
- Vague milestones
- Negative feedback from management
- Lack of openness and participation in war team
- Inadequate testing resources
- Inefficient testing techniques
- Poor quality documentation reviews and review scheduling
- Poorly organized code for localization
- Confusion over the role of IPG
- Confused ownership of build process and OAK
- Deficient quality of external acquisitions
- Poor bug tracking tool

What Should Be Done Differently?

The discussion of what worked and what didn't leads naturally to a set of recommendations as to what to do differently next time. These are described in this section. Those aspects of the process that were considered to work well are not repeated here.

Specifications and Product Definition

One of our obvious weaknesses in the DOS 5.0 project was the inadequate initial definition of the original product. We didn't spend enough time early on thinking about the required feature set, nor about the details of how we would implement it. Next time we should spend more time up front refining the product plan, with the expectation that this will reduce the need for ad hoc additions later.

It is important that the product definition phase include broad participation from the various product constituencies. The goals need input from more than just Development and Program Management. Results from usability tests (run by User Ed) and requirements from customers (collected by Product Management) should be used in the product definition.

One of the successes of the DOS 5.0 was the clear and consistent theme or overall goal. Naturally, every project should have a goal that can be stated as clearly and simply, and a sharp understanding of the target audience. Every item in the feature set should relate to the goal in an obvious way.

The product specification should include not only the development spec and schedule, but plans from Testing, User Ed, and IPG as well. The needs of these groups should be integrated into the schedule from the beginning, meaning that their plans should be complete at the same time the development spec is complete.

The specifications should be maintained online, under source code control. Updates should be made and distributed regularly. Keeping the spec accurate will help reduce confusion as the product evolves.

We need to keep the dual nature, retail and OEM, of our product in mind. Remember that our deliverable to the OEMs needs to be of the same quality as the deliverable to the end user. We need to spend time make the OAK package as accurate and as usable as possible.

Schedules

Improving the quality of scheduling requires a combination of attitude adjustment and new techniques. We will create better schedules when we have a common ground for building them, and a common goal in their creation.

The development team needs to lose its fear of commitments. We need to recognize the necessity of defining targets, and accept the importance of making the targets once they are committed. Acceptance of commitments will tend to enhance the team's sense of ownership of its product, and reinforce a sense of shared responsibility.

Willingness to accept commitments is the basis for building better schedules. The first block to lay upon that cornerstone is a shared understanding between Program Management and Development of the meaning and use of time estimates. Developers need to produce specific estimates that program managers can use; the exact mechanics of estimate creation, padding, task breakdown, and so on, can be negotiated

on an individual basis. All developers should recognize, however, that the estimates they provide can and will be turned into commitments at some time.

A well-constructed schedule requires well-defined interim milestones. We are familiar with the usual markers of this sort -- code complete, alpha release, beta release -- but we need to make sure they are meaningful and achievable. We should also take the time to acknowledge them when we reach them -- celebrating interim achievements will help foster a sense of momentum and progress, as well as providing opportunities for team building.

Schedules, like specifications, should be kept up to date. As we make progress on a task, we can improve the accuracy of our estimates; schedule documents need to be updated and circulated and milestones adjusted on a regular basis to reflect any changes. Modified schedules should be distributed widely, so that no one is surprised by changes. Overall schedule reviews should themselves be written into the project schedule, so that we can do periodic sanity checks in a measured atmosphere, rather than a frantic one.

To help ensure the accuracy of our schedules, we need to develop a series of metrics to aid in prediction. We have raw data available from the beta and bug fixing phases of the DOS 5.0 project. We should assemble statistics on bug fix rates and bug incoming rates, look for patterns, and use the information as a benchmark for similar phases of future products. We should investigate what other metrics we can develop from the data we have, but the bug rates are easy to extract and analyze, so they make a good starting point.

We need to avoid situations in the future where critical path items depend on a single irreplaceable person. We were at times in the course of DOS 5.0 confronted with the possibility of serious schedule slippage if a key person became seriously ill or was in a serious accident. Fortunately, that never occurred, but we should protect ourselves by ensuring that knowledge of key areas is shared. This isn't meant to require redundant specialists in the development staff, but for any key piece there should be at least one other person who knows enough about the internals of that piece to come up to speed quickly in an emergency.

Communication

While informal communications among individuals worked well during the course of DOS 5.0, more formal communications among groups were sometimes lacking. This resulted in wasted work, duplication of effort, and schedule delays. Things improved later in the project, in particular when war team notes were distributed widely. This practice should be routine from the project inception in the future.

Indeed, the war team itself should be constituted at the beginning of the project. The war team should be large enough to represent all constituencies, but it is important to keep it a small and well-focused group, so that discussions can be kept on track, and meetings can be scheduled easily. Only one representative from each constituency should attend the war team meetings on a routine basis, except where there is a clearly recognized need for more. On the other hand, the war team should strive to be an open forum where all team members can bring issues for discussion. We must avoid the image of distance and mystery that surrounded the DOS 5.0 war team in its early days. As a direct reminder of this, and to avoid objectionable military associations, the war team is now referred to as the Forum.

In addition to the Forum meetings for the management group, there should be meetings for the entire team from time to time, to disseminate overall status and reinforce the sense of common purpose. These occasions can be used to help the team see the bigger picture, and clarify the competitive environment and strategic vision the product is part of. The meetings conducted by Product Management in the late stages of the DOS 5.0 project, where the marketing and advertising strategies were presented, were good

examples of this approach. Celebration of interim milestones, as discussed previously, is another vehicle for these group meetings. Another important occasion is the project kickoff meeting, where goals can be defined, and roles and responsibilities clarified.

Part of the success of individual contacts and communication in DOS 5.0 lay in the relatively small size of the team. With both large and small teams, however, there are steps that can be taken to enhance communication at the individual level. It would be helpful to publish "expert lists", defining who are the specialists in specific areas, so that, for example, documentation writers would have a quick way to find out who to direct questions to. We should exploit the specialist knowledge we have developed to increase everyone's familiarity with our customers' environment through seminars and training sessions. We should cultivate opportunities for team building within and among the component teams of the project.

Documentation

Many of the problems experienced by the documentation team will be alleviated by improved communications and specifications. There are some other specific steps that will need to be taken to improve the quality of their contribution to the product.

First and foremost is the need to develop a more effective technical review process. This process should be a topic of a separate study and discussion between User Ed and Development. Some ideas to be considered are distributing and reviewing material on-line, developing automated tools to ensure changes are made correctly, and extending the review period so that reviewers are less overwhelmed by the volume of material.

The programming writers need to be integrated into the team in the same way as the end user writers. Although in our case some of the programmers' documentation is not included in the product, it is still a crucial item in a new operating system release, and needs to be delivered on the same schedule. In addition, there is a very important piece of programmer documentation which is part of the project, namely the OAK guide. User Ed needs to take ownership of the OAK guide, and make it an accurate, usable document. The OAK guide is the face of the product to our OEM customers, and as such needs much more attention devoted to it than has historically been the case.

Finally, User Ed can assist PSS by compiling PSS technical notes in advance of product release. As we encounter bugs, and in particular as we choose to postpone bugs, we can identify difficulties that our customers are likely to encounter. If we cannot adequately discuss remedies in the product documentation, the information can be tracked and written up by User Ed, so that it is available to support technicians and customers when it is first needed.

Testing

As discussed previously, there was widespread agreement that Testing needed more resources to do its job more completely. The obvious recommendation is to increase resources, both personnel and equipment. Since hiring qualified testers is difficult, recruiting efforts should begin well in advance of anticipated need. Equipment acquisition needs to cover a broad spectrum, but special attention should be given to obtaining older hardware, hardware made and sold overseas, and a wide range of third party add-in equipment, such as tape drives, removable disks, video cards, and so on.

The DOS 5.0 test team made considerable use of temporaries, with mixed results. Although some of the temps made valuable contributions, the overall feeling was that we would have been better served by additional fulltime heads in their place. We should avoid relying on temps for crucial testing needs in the future.

Likewise, we should avoid over-reliance on external testing houses. They did make some contributions, but didn't fully meet our expectations. They were not a comparable substitute for adequate application testing on our part. Again, the expenditure would be better spent building our own fulltime team.

The existing team needs to do its part to enhance its effectiveness, as well. Use of automated testing techniques should be increased. Quick regression tests that developers can use before they check in changes, comparable to the OS/2 model, should be developed. Component testing needs more emphasis, and testers need adequate time to explore creative ways to test boundary conditions, abnormal failure conditions, and system stress conditions.

Developers also need to put more time into upfront testing and regression testing. White box testing, debugging traps, and debugger walkthroughs should be a routine part of each developer's work. When code is checked in and bugs resolved, developers need to describe adequate test plans for all bugs. Developers can help testers better focus their efforts by identifying up front parts of the project expected to be bug prone.

To avoid the lab maintenance problems that plagued us during DOS 5.0, Testing should schedule time for lab cleanup. A checkout system could be put in place for occasions when machines need to be reconfigured, to ensure that everything is returned to a known state. Machines that need repairs should be set aside in specific areas, and PC Repair notified promptly. Machines that are beyond repair should be disposed of. Storage space for shipping boxes that must be retained should be organized.

Finally, the relationship of Build to Test and Development must be clarified. The builder and his product are really within Development's responsibility, so the builder should report to a development lead. Development should take responsibility for the integrity of the build, although Testing should continue to verify it with their smoke tests. Development should always be aware that the source tree is the product, and ensure that it is kept in good repair.

International

The domestic team needs an understanding of the international users' needs, and not rely on IPG to provide that information. Gaining that understanding is a challenge for the domestic team. We must develop the contacts and expertise needed to analyze and address the international DOS marketplace. A specific effort should be undertaken by Program Management, Development, and Testing.

To aid the localization effort, the development team needs to clean up the inconsistencies in the organization of localizable code. Some progress has already been made in consolidating the messaging model used by the DOS utilities, but we need to make sure we do as much as we can. In general, Development needs to work with the mindset that English is just another foreign language, albeit one that doesn't require translation, and to regard IPG as another OEM customer for the DOS sources.

IPG, for its part, should be proactive in pointing out problems in localizability or international support, and help the domestic team build the expertise required to make sure DOS becomes a great international product. Early intervention on the part of IPG will help considerably in ensuring problems get addressed in a timely fashion.

External Dependencies

External dependencies led to unanticipated problems. We should continue to seek opportunities to leverage external resources, but we need to do a better job of managing them. For internal groups, Program Management needs to take the lead in establishing contacts with their peers in the other groups, making sure our needs and schedules are well understood and up to date, and negotiating appropriate priorities for our needs. Development needs to communicate clearly their needs and expectations, both to their counterparts in the other groups, and to Program Management on both sides. Where it proves impossible to obtain adequate priority, we need to consider alternative sources for the particular component.

For components coming from outside the company, we need to do a more thorough job of reviewing the deliverables and specifying acceptance criteria in advance. Every contract should include well-defined acceptance criteria, focused on features, performance, bug numbers, code size, and whatever other measures are appropriate. Testing should define specific acceptance tests that must be passed before delivery is accepted, and Testing must sign up to run these tests. IPG should review all code for localizability, and any problems should be addressed before delivery is accepted. Development should review code for adherence to reasonable coding standards and quality, and User Ed should ensure that interface design is sound, the component is usable, and that they can adequately document it in the time frame, all before formal acceptance. This process will take more time upfront, but will reduce costly errors and time loss later..

Tools

While tools were not a big problem over the course of the DOS 5.0 project, there are a few areas where improvements could be made. The most important would be to replace the RAID bug tracking database. Fortunately, the changeover to the SQL-based version of RAID has addressed many of the performance, capacity,and reliability issues we experienced, but we should give more thought to what we require from a database of this sort, and encourage the maintainers of the RAID tool to enhance their product to meet our needs.

Developers could benefit from a database to manage source code libraries, and preserve common routines and techniques. At present, making use of existing code requires folk knowledge of where the code can be found, or at least of who to ask. An organized database of source routines could help reduce unnecessary repetition during development, and through active code sharing, increase productivity and reduce maintenance costs.

Summary of What Should Be Done Differently

- Spend more time on product specification
- Elicit broad participation in defining product
- Define a clear goal and audience for product
- Include test, doc, localization plans in specification
- Keep specs and schedules up to date
- Define clear interim milestones; celebrate them when achieved
- Distribute specs, schedules, meeting notes to all affected parties
- Analyze bug fix rate and incoming rate data to provide baselines for future estimates
- Establish Forum group at project start
- Distribute Forum meeting notes broadly
- Hold a kickoff meeting for the entire group, and occasional meetings through the project
- Publish expert lists and organize seminars

- Develop better documentation review techniques
- Integrate programming writers into documentation team
- Transfer responsibility for OAK doc to documentation team
- Augment testing resources
- Develop more efficient testing techniques
- Develop and use white box testing techniques
- Augment international expertise in domestic teams
- Clean up source tree to facilitate localization
- Apply rigorous criteria to external acquisitions in advance of acceptance
- Enhance contacts with internal groups we depend on.
- Transfer build function to Development
- Investigate database tools for bug tracking and source libraries

Appendix - Raw Comments from Post-Mortem

Attached is the record of comments made at the post-mortem meeting. It is organized around the three basic questions, with the comments made by functional area in response to each questions. The overall post-mortem report was based on these comments. The notes were recorded by Greg Tzinberg.

These notes are transcriptions of individual opinions as they were expressed, and so will at times contain contradictions or transcription errors.

Things that worked well.

Overall
* Focus on product quality, (improve on DOS 4).
* Product compatibility with previous versions.
* There was a clear focus on the objectives for the product.
* The team's behaviors supported the stated goals on quality.
* There was a built in measurable, i.e., the size of the kernel.

Program Management
* The process was driven by usability and the needs of the user.
* Listened to feedback
* Regarding the Beta Program-People perceived it as "in sync".
* Implemented a direct feedback program where discs were sent back to team.
* Bug fixes-became a team effort, everyone participated.
* People worked together and helped each other solve problems.
* There was an effort made to increase the group's resources.
* They looked at the project from more than one perspective.
* They had documentation "walk throughs".
* Instituted a program of "postponing bugs".

War Team
* Provided focus, leadership, decision making and follow through.
* Had an interdisciplinary make-up.
* People had input and therefor greater buy-in.
* People identified the War Team as a forum were things could be decided.
* People felt that they could rely on the War Team.
* When the War Team's notes were distributed, that was regarded as very helpful.

Testing
* The change from OEM to user based product.
* Focus on compatibility.
* Introducing the concept of specialized tests for various functions.
* Having adequate resources.
* Having a huge Beta test.
* Having the users sending back the un-install disks.
* Beta reports.
* Instituting a smoke test procedure.
* Smoking immediately after builds resulted in less down time.
* The relationship between Test and Development became very positive.
* Group was not overly sub divided, resulting in easier flow of information.
* Good turnaround time on test information back to developers.

MS7021005
CONFIDENTIAL

* The project tree was very stable, the build environment was improved by having bug reports sent over e-mail.
* The internal mini-tests "saved our butts".
* There was a responsive test team on networks.
* Having a set build schedule helped to focus and structure the project.

User Education
* We responded well to the shift in product focus, when the product was geared toward retail.
* Used documentation to fix the product.
* Positive writer/tester relationship.
* We were encouraged to offer our unique perspective.
* UE kept up a high morale in spite of the stress.
* UE reviewed all the text that would be seen by end users.
* The tech review walk throughs were very helpful.
* Having the role as "user advocate" helped us define our contribution.
* We pursued the developers in spite of resistance.
* We were strong advocates for "Network Install".

International Product Group
* Worked in close tandem with the developers.
* We were given good support from the developers.
* Structure of the schedule was stable, reliable.
* The quality of the documentation was high.
* The code was clear.
* The product was easier to de-bug.
* IPG raised the consciousness of others re. localization issues.
* IPG was willing to "live with" the development team.
* IPG was pro-active in seeking the information it needed rather than waiting for others to provide it.

Overall
* People showed commitment by being flexible.
* The team developed positive working relationships.
* They had pride in the product.
* The smallness of team contributed to it's success.
* People cared beyond their specialty.
* The specialists were of high caliber.
* The end user perspective was well advocated with consideration given to the many different levels of end users.
* Customer satisfaction became a high priority.
* The Compuserve Serve Forum was very helpful.
* Direct interaction with end users provided valuable information.
* People were accountable for completing their tasks.
* Few of the bugs were from sloppy code.

Things that didn't work.

War Team
* War Team began as a mysterious body, people did not know enough about it.
* Became dictatorial.
* Participation was not invited at first.
* Decisions were put off, or cycled too many times.
* There was a lack of follow through.

Development
* The "pep talks" they were too negative in tone, uninspiring and contributed to low morale. People felt attacked and unsupported by upper management.
* Milestones were not set and when met went uncelebrated.
* There were no opportunities to recognize the contributions of individuals.
* The interim goals were not well defined.
* Information that managers knew was not shared (for example, the milestones).
* Build 409 went on and on.
* Some goals were seen as unrealistic.
* The upper management was unprofessional in the way that they responded to goals that we missed.
* People did not "own" their dates. Accountability became an issue.
* Over half of the work that people did was unanticipated by program management.
* Much of the work was "reactive", especially in the last 15 months.
* There was blaming, for example, upper management blamed the developers of not knowing how to work hard.
* Development should have realized that a code change impacts the work and schedules of most of the other people on the project.
* Many parts of the project did not have a "spec".
* It was overwhelming to receive 150 pages from U.E. for tech reviews.

Program Management
* Not knowing the quantity of work involved in the project.
* The initial design phase was too short.
* When the project was re-designed it should have been noted or marked.
* Upper management does not understand the "reactive" quality of the development process.
* There was a lack of clarity regarding the priorities, too much reactivity.
* Upper management vetoed schedules.
* The schedules were too general.
* There seemed to be a lack of understanding between program management and upper management.
* Program management over designed, (designed Cadillac when V W would do).
* The ownership of the schedule remained unclear. There was unclarity regarding roles.
* Building a schedule from the work estimates did not work.
* Being dependant on outside resources did not work.
* Communication between developers, program management and upper management was poor.
* There was no coordination between schedule changes and spec changes.
* Developers often had the impression that they were not heard during meetings.
* "Quick" decisions (example, "Install") did not work out well.
* The resources devoted to test were too limited, the priorities were unclear.
* At times the focus on quality left the end user out of the picture.
* At times developers did not solicit enough input, (example, "Un-delete").
* There was too much dependence on one person, what happens if a bus hits 'em?

Testing
* We were out of touch with the needs of end users.
* There were problems with the build process.
* Later additions were made without considering the impact on test.
* We needed more access to older machines.
* There was too much emphasis on bug regression.
* There was a lack of clarity re. ownership of the build process.
* The test lab was poorly maintained.
* There was a lack of foreign machines.

User Education
- There were problems in communicating schedules, and specs.
- We were not included in the decision making loop.
- There were conflicting messages from developers and program management.
- UE not well utilized in the design phase of the project.
- Tech reviews were of poor quality.
- There wasn't enough feedback between the writer and the developer.
- Tech reviews were not included in the schedule.
- OAK wasn't wanted by anyone.
- There wasn't enough fun.
- Developers needed more time to review the docs.
- U. E. was not well integrated into the "culture" of developers.

International Product Group
- It was difficult to track the doc changes.
- Re-translating the spec 3 X's didn't work. (Very expensive.)
- IPG not on War Team for 3 months.
- Messages were scattered throughout the "tree".
- Test was not testing the international features.
- Lack of follow through on tracking changes.
- We were running out of memory on set up.
- IPG did not pursue vital information enough.
- Beta needed to out earlier.
- Testers did not understand our needs and requirements.
- There was insufficient testing done on international machines.
- IPG not involved early enough.
- There was a "disconnect" at a higher level of management re. what IPG responsible for and what Dev and Prog Mgt was responsible for.

What would you do differently?

Development
- Write better specs. Spend more time on writing them.
- Have access to international and other machines.
- Define planning and goal definition.
- Build in an opportunity to do mid-course evaluations/corrections.
- Build in a periodic plan review with enough time scheduled to accomplish them.
- Develop a set of questions/criteria for product.
- Make it possible for anyone to call for a review process of the project.
- Developers need to be more assertive.
- We need to keep Prog Mgt better informed.
- Pass around status reports and schedules with milestones from all the groups.
- Update the schedules on a monthly basis.
- Develop a quantifiable method of measuring status.
- Improve the quality of the status meetings.
- Improve communications so that they are two way rather than one way.
- Clarify when requesting a work estimate or a commitment to a certain date.
- Institute "white box" testing. Schedule needs to allow developers to test.
- Improve the quality of communications to Test.
- Respond realistically to target dates.
- Become more involved in the inception of the product. More involved in the creative process.

* We want Prog Mgt to trust our time estimates.
* Eliminate the dependencies outside the group where possible.

Program Management
* Spec and test plans prior to code.
* Include "white box" testing.
* Document decisions, communicate them to entire group.
* Change the name War Team to Forum.
* The Forum needs to be established at the beginning of a project.
* Communications should include all relevant members of the team.
* User Ed should own the OAK doc.
* Celebrate hitting the major milestones.
* Team should be made aware of and help solve major milestone slips.
* Develop a test plan for the OAK.
* Two people for each critical piece of the project.
* Program Mgt should "pad" schedules.
* There should be a major "blood letting" on the subject of schedules.
* Schedules need to be broken down by tasks.
* Eliminate outside dependencies wherever possible.
* Define the audience of the product.
* Give the OAK to U E.
* Review the process of third party acquisition of software.
* Have IPG review the adequacy of the third party software.
* Create a negotiation document.
* We need to improve our relationships with our internal dependencies.
* Sources need to be accessible.
* The build function needs to be moved to development side.
* There needs to be additional conduits for communicating with various groups.
* There needs to be a localized version in German.
* Establish a DOS alias.

Testing
* We need to improve our methodology.
* We need more resources.
* Improve our ability to track bugs on data base.
* Earlier involvement in product development.
* Spec should include the test plan.
* Testers need to challenge the developers more "up front".
* We want clearer directions from the developers.
* Testers should have an expanded role.
* ISV's need to run more tests.
* There needs to be cross training.
* The test cycle needs expanding in order to relieve regression testing.
* Assign person to maintain labs, configurations, procedures etc.
* Educate testers as to external network issues. "How do customers use our products?"
* Have testers train the developers on Novelle.
* Solicit requests for hardware.
* Do some "clever thinking" on bug regression.
* Automate the test process more.
* More resources, equipment and heads.
* Develop better "bug fix" documentation.

User Education
- Negotiate the OAK.
- Engage in continuing education re. the product.
- Have programming writers on the end user team.
- UE should own usability testing.
- Define the audience of the product.
- Focus groups.
- Keep a record of trouble areas for PSS.
- Develop a proposal on how to improve the tech reviews.
- Have people recognize U E 's user advocate role.

International Product Group
- Look at the "tree".
- Become more pro-active.
- Clarify responsibility that development and test has for localization.
- Ask upper mgt to clarify the ownership issues.
- Treat English as a foreign language.
- Have IPG sit in and own the IPG issues on the DOS team.
- Write for universal English rather than American English.
- Educate the developers as to how to develop with International in mind.
- Develop a charter as to who does what.
- Research features that may be needed in international versions of the product.