Good morning and welcome to the sessions on Portable SAS.

I. INTRODUCTION

My plan in this session is to give you a little insight into the porting process we have gone through at the Institute. Ours is a unique experience in many ways: the size, complexity and management style are not out of a textbook. Our goal was simple: to implement Institute program products on the Digital VAX, DG MV and Prime 50 series computers. Over the next several months, we will refine the process and begin porting the product to other machines yet unspecified.

To give insight, I will review classical portability concepts and present two 'new to me' management concepts that I call 'Parallelism' and 'Developmental Flexibility'.

Let me note first however, that immediately following this presentation there will be a panel discussion by four Portable Systems staff and myself to discuss the progress we have made over the last couple of months and where we will be going over the rest of the year. In addition, there will be a BOF this evening between 6:00 and 8:00 where we will be available to answer questions on the process of the product.

For the next part of my introduction, I am going to provide a little history of the portable system, some statistics and a view of the way we are trying to do things.

The portable system project started in February 1982 with the goal of putting the SAS® System on minicomputers as quickly and efficiently as possible. We have essentially accomplished that goal. The Digital system is in production, the Data General will be there shortly and the Prime will be going into test in a few months.

[ SLIDE: SYSTEM STATISTICS ]

The entire effort will be completed by the end of this year. We will then have released 5 products on each of 3 machines -- a tremendous feat. We will say more about this during the Panel Discussion and BOF.

The system is large and complex. About 450,000 lines or 15 mb of source code in approximately 2000 modules. The Data Step was redesigned and implemented based on the 1982 User's Guide. The procedures were mostly converted from the IBM although some have been completely rewritten. For developing the system, we are now implementing both a filter, to insure that the code coming from the developers is portable (something not easily assured) and a source manager to ensure that all changes are made everywhere they are needed. The system is being developed on the Digital VAX and then 'ported' to the D.G. and Prime systems.

[ SLIDE: DEVELOPMENTAL PHASES ]

We will now move into the porting process and see how we developed it at the Institute. As you probably are aware, the classical definition of the porting process involves moving software from one environment to another with significantly less cost than developing it completely from concepts on the second and succeeding systems. This process involves reducing the cost of the four phases of any software system: Definition, Development, Testing and Maintenance.

However, the classical concepts apply only partially to our project for three reasons:

1. We tried to move from one environment to three simultaneously. This has resulted in three porting efforts all rolled into one and adds several layers of difficulty to the process which are seldom discussed in the literature.

2. Part of the system, the data step, was completely rewritten. However, moving from the Digital to the other minicomputers follows more traditional porting procedures.

3. The system is part language, part application and part operating system. This is not a problem in the definition, but refers more to the literature that usually treats only one of those conversion types giving a rendition of the unique characteristics of the problem depending on whether it is a language (which we are), an application system (which we are) or an operating system (which we have elements of) and not dealing with porting all three simultaneously.
In addition, when people discuss portability projects, they discuss either the obstacles to portability or give guidelines for portability projects -- both of which are needed and I will review them -- but they seldom deal with the actual process of 'doing it'. I will deal with that later by discussing two concepts we have developed for managing software projects.

The challenge to the portable systems developer is to minimize machine dependence while maximizing system efficiency and flexibility. The obstacles the developer faces can be related to:

[SLIDE: OBSTACLES]

To handle these obstacles, we can do one of the two things:

1. design to use them realizing the non-portability aspects
   or,
2. design around them (i.e., for machine independence)

The trade-off is between efficiency and flexibility. We choose to write for machine dependencies since we are operating system oriented thereby decreasing flexibility but increasing efficiency potential.

[SLIDE: MACHINE DEPENDENCIES]

Machine dependencies are inherent in porting an operating system because operating systems rely on the hardware for:

- Interrupt handling (system calls, traps)
- I/O (device drivers, peripheral control)
- Memory Management

but, these segments can be written in PL/I, at least initially to maximize flexibility and portability.

About 6% of our code is machine dependent. We expect this to increase to 10% as we write more code in host assembly languages to improve performance.

To summarize the classical approach, we use these guidelines:

[SLIDE: GUIDELINES FOR WRITING PORTABLE SYSTEMS]

1. Determine the system you want to port
   -- base
   -- incremental enhancements

   This may sound trivial, but as we will see, this is not the usual situation; in our project, several major features have been added since its inception.

2. Identify, design, document and maintain standards. This can be done and is critical for ease of development.

3. Develop, review and select machines.

4. Select language and the common subset -- standardize the subset.

   This is not easy -- programmers want to use what is available and what they know -- use filters to ensure it happens.

5. Identify machine dependencies -- have 1 person know each machine intimately, and have 1 person familiar with all machines.

6. Introduce procedures and tools to ensure standards are followed; use filters, verifiers and source code managers. This requires $5 and ensures $2 and $4.

7. Design the system for change and redesign. This is not part of of the normal approach, but necessary for practical development projects.

8. Develop internal and user documentation simultaneously and continuously -- it is a never ending process.

9. Develop and maintain test programs during all stages of development.

10. Understand the constraints of hardware, O.S., languages and PEOPLE. (Are they idea or follow-through oriented; are they manager or implementor, what is their level of perfectionism and their depth of knowledge of the tool and the system.)

Certain projects take a certain amount of time no matter how many people you throw at them. If it's a single person project, then give it to the developer and get an estimate of how long it will take -- know your multiplier factor and add that to the list of constraints to when you can ship the system.

That is the classical approach. We followed most of these ideas, some more closely than others, but all of them at least every now and then.

However, we are constantly behind schedule and I have spent a lot of time wondering "Why?".

The project is big;
It is complex
All the guidelines weren't followed -- but they could not be given what we had to do, particularly in defining the system. So, why the delay? Because we didn't include in our estimate, the concept of moving target development. Our effort was directed at getting the next up to date product into the field at the earliest possible moment, but the product was (and still is) being defined. This is as it should be in a fast-moving development environment. Thinking about this has led me to two concepts of managing such projects which I want to share with you. They should give some more insight into some of the delays we have had and may be helpful in your own software development efforts. I do not profess to be the originator of these ideas nor are they fully developed in my mind. I am very interested in your thoughts on them and your experience in developing systems because we are looking for ways to decrease our development time. I call the two concepts 'Parallelism' and 'Development Flexibility.'

**Parallelism** is a concept that says to get the latest ideas into the field in the shortest amount of time we must schedule the different stages of development in parallel, not serial. This is analogous to parallel vs. pipeline processing. In hardware terms, given a CPU of a given MIP rate, the only way to speed processing is to use several processors to perform simultaneously all independent tasks and later, glue them together. This requires another level of supervision, but can greatly speed the process. I was looking at communications processors and most had 2-4 MHz capacity. One had 50 MHz capacity and was only slightly more expensive, much more flexible and could handle many more lines than any other system. It did not make sense until I understood that it used 25 7 MHz processors working in parallel and managed by other processors. And what is a VAX 11/782 but two 780's closely linked. The concept has been used widely in the computer industry, but not in development of software products. In software development, we want to work serially but can never seem to meet the demands of our users.

In order to produce software products quickly and satisfy user demand to give them what we have as soon as we have it -- we need to parallel process the various development stages more fully. Let's look at management of portable products from a higher level. Up to this point, I have talked about the software development effort. This is only one of at least 6 efforts that have to be managed.

**The others are:**
- Documentation
- Testing
- Education
- Marketing
- Distribution

The desire of developers and support personnel is to perform these efforts step-wise. This is the simplest and easiest to understand concept. It is that great desire to learn to crawl before walking and walk before running. Very natural, probably rational, but not always practical in the age where getting a product to market six months before the competition can be a 'make or break' proposition or the delay can cause a lot of developer grief and user dissatisfaction.

At the highest level, we can look at the process as:

1. **Alpha** - Ideas formulated + Release features defined
2. **Beta** - Design specs written (reviewed)
3. **Prod** - Documentation development + System development
4. **Maint** - Final documentation written + Q.A. Testing + Education materials developed + Marketing + Distribution + Alpha test
5. **Beta test**
6. **Production**
7. **Maintenance**

Again for comparison, the classical approach is to design the system and serially go through all the development stages one by one until we get to the alpha test. When that is complete, we begin developing the beta test -- and so on, around and around until we get to a new version and start at the alpha level again. This applies to all changes from the most trivial to the most profound, including fixing bugs. This may be the only sensible way given our own limitations in dealing with complexity, however, it is not what we have done in the portable project.
The difficulty with this system comes when new ideas need to be incorporated after old ideas are defined. When you want to market the program as soon as the development of the first idea is completed but incorporate as much of the second idea as possible in the product. Although this is a difficult proposition, it is not impossible.

To get new ideas out faster, we need to overlap more of the subproject or system phases so that they all start early and delay only projects that do not have anything that can be worked on. This is particularly true of the support projects. This changes what is expected when. People will resist this overlapping, but with proper emphasis, the idea can be used to improve development time.

A major effect of this is that managers will be busier with each project and will not be able to handle as many as major projects. But projects will get done in significantly less time. The trade-off is not linear and I do not know what the trade-off curve looks like.

This slide shows from top to bottom, the major components of a simplified development process. The circles highlight the fact that new ideas are formed, designed and implemented before the prior release was distributed. This is why we want to ship but to do so, we need to ‘pull back’ the start of the last phases in the flow.

Let’s expand this picture a little.

This slide shows where we have reduced the time between development and distribution and what needs to be done to shorten it to its minimum.

If we identify those items that are independent from the development phase, we can ‘pull’ them back to be completed when development is completed. This may ‘push’ development out a little, but the pull time, being accomplished mostly by independent people will be larger than the ‘push’ time. The result is that the release date will be sooner.

This requires a new non-textbook way of doing software development. We were forced into doing this by the need to get the product to market when the magnitude of the task was greatly underestimated and the need for adding totally new features was not even considered.

The above flow of work considers only one machine. When we went to the second machine, we learned that there are many common components, but there are also many non-portable components. This will occur with each new machine. In our efforts, we have found sufficient differences to consider each new machine an almost unique porting effort. The portable code needs to be made more portable. We hope the magnitude of differences will shrink; however, for at least two of the new systems we’re looking at, this is not true. The effort required to change the portable code will be large. One of the reasons this occurs is because standard PL/I Subset G implementations are not standard and the missing components are critical to our effort. The latest is a compiler without REPLACE or UNALIGNED Bit Variables.

It is also clear that to manage this project is a tremendous task. When we incorporate all distribution levels for each of 3 machines and 5 products, this simplified chart turns into a piece of diagonalized graph paper.

These slides also do not show the iterative nature of the porting process. We have found that as we add new features to the system, we have to redesign both portable and machine dependent code. This occurs even though we may have designed the features in at the beginning, but did not consider all of the implications. We invariably missed “some detail” that required redesign. This is greatly compounded when the new feature was not even considered in the original design, such as the Display Manager. With this feature, significant sections of code had to be designed and added to the system. These activities have led to the notion that the porting process is an iterative one. This is reflected in the charts with the new features being added at the top for each release, but it also pervades the entire process particularly between releases when we are porting to the second and succeeding machines.

Another deduction we can make from this discussion is that parallel project management requires the utmost in communication, coordination and cooperation. Without some, it will not work and without a lot, it will not work well.

To summarize: Parallel management means to develop logically, sequentially related components of a project as simultaneously as possible. This requires considerably more management and increases project complexity but provides the shortest possible time between conception and production.

The second idea I would like to discuss is level of developer flexibility during the life of the project. The idea here is also to attain the highest speed of development while maintaining control over product quality and integrity.

The amount of independence you give a developer/programmer will vary greatly depending upon your organization. However, within any organization, the amount of flexibility given to a developer will directly influence the speed of development and the integrity of the system. The
relationships between speed of development, developmental flexibility and system integrity are shown here:

When I say 'Speed of Development', I mean the time to get a feature implemented to the point where it can be turned over to internal testing. In this phase, I also include effects of high stress situations.

'Developmental flexibility' is the amount of independence a developer is given to design, revamp, write, insert, implement, or change code.

'System integrity' is the state of correctness of the system.

From the non-linear and direct (and inferred not measured) relationships shown in the slide, we can see that there is a speed of development for a given level of integrity we desire and a given amount of flexibility we wish to allow our developers. As we proceed through the development cycle, we want to maximize both speed of development and system integrity. However, these are inversely related and both depend on developmental flexibility. Most software development efforts stress minimizing flexibility and ignore the issue of development speed, implicitly implying that high control is the fastest way. This is not necessarily so.

There is one additional complicating factor. There is a relationship between integrity, speed of development and amount of time needed for code correction (that is, fixing bugs). As speed of development increases, the amount of time that will be needed for code correction also increases. This relationship is present but unclear from our work. It appears to be flatter than the speed of the development curve at slow speed but rises sharply at high speed or under pressured development.

Thus, we have to make some choices in the management of the project. For our project, we have only begun to understand these choices. Looking back, however, we can see choices that would have improved both the integrity of the system -- which is high as it stands -- and the speed of development -- which is good.

Of the four basic development phases, the actual development has been by far the longest. (See chart below.) We could have minimized development time by minimizing development flexibility when defining standards, the problem and the solution. These should have been done in a very highly controlled manner to ensure completeness and correctness. When development started, each developer was and should be given complete flexibility within the established standards and problem definition. However, as development changes to code correction, there is a need for greater integrity in the system. Changes to the system need more centralized review and developer flexibility should have been reduced. Developers may have to wait on tests and acceptances to their changes to the system, but this ensures system integrity. When the system enters the maintenance phase, development and system changes become difficult and time-consuming because of the need to maintain absolute system integrity. Finally, the control for this integrity must be centralized. This procedure will lead to maximum development speed.

<table>
<thead>
<tr>
<th>SLIDE: FLEXIBILITY LEVELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility Levels to Minimize Project Development Time</td>
</tr>
<tr>
<td>Flexibility</td>
</tr>
<tr>
<td>Low</td>
</tr>
<tr>
<td>High</td>
</tr>
<tr>
<td>decreasing to</td>
</tr>
<tr>
<td>Medium</td>
</tr>
<tr>
<td>Low</td>
</tr>
</tbody>
</table>

To summarize: We have learned that the classical guidelines to producing portable projects are insufficient in several major areas. First, they do not allow for changes throughout development and they greatly increase development time. To reduce development time, we need to implement parallel development of as many of the development stages as possible -- which is nearly all -- and to allow for varying levels of programmer flexibility during the development process.

And, with this I conclude this presentation. I look forward to your comments and your thoughts on these ideas. Thank you for your attention.
SLIDES FROM "MAKING SYSTEMS PORTABLE"

TOPICS OF DISCUSSION

CLASSICAL PORTABILITY CONCEPTS
MANAGEMENT CONCEPTS:
PARALLEL DEVELOPMENT
FLEXIBILITY CONSIDERATIONS

SYSTEM STATISTICS

NUMBER OF PRODUCTS
SIZE OF PRODUCTS

DEVELOPMENT PHASES

1. STANDARDS DEFINITION, PROBLEM DEFINITION AND ANALYSIS
2. DOCUMENTATION AND SOFTWARE DEVELOPMENT
3. TESTING AND CODE CORRECTION
4. MAINTENANCE

DIFFERENCES FROM CLASSICAL CONCEPTS

- TO 3 ENvironments
- PART REWRITTEN FROM SCRATCH
- PRODUCT IS BOTH OPERATING AND APPLICATION SYSTEM

OBSTACLES:

- ARCHITECTURAL
- OPERATING SYSTEM
- LANGUAGES (COMPILER)
- PERIPHERAL DEVICES

MACHINE DEPENDENT CODE

- INTERRUPT HANDLING
- I/O
- MEMORY MANAGEMENT

GUIDELINES FOR WRITING PORTABLE SYSTEMS

- Determine the system
- Develop standards
- Select machines
- Identify machine dependencies
- Introduce tools to run standards
- Design for change
- Develop documentation
- Develop test programs
- Understand the constraints

Remember... You cannot trade people for time & you will need more time than you think!
OTHER EFFORTS THAT HAVE TO BE MANAGED

- Documentation
- Testing
- Education
- Distribution
- Installation

PARALLEL PROJECT MANAGEMENT YIELDS:

- Complexity
- Complexity Requires:
  - Communication, Coordination
  - and Cooperation

FLEXIBILITY LEVELS TO MINIMIZE PROJECT DEVELOPMENT TIME

<table>
<thead>
<tr>
<th>Flexibility</th>
<th>Phase</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Problem definition &amp; analysis</td>
<td>Centralized</td>
</tr>
<tr>
<td>High</td>
<td>Early development</td>
<td>Decentralized</td>
</tr>
<tr>
<td>Medium</td>
<td>Late development</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>Testing &amp; code correction</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maintenance</td>
<td>Centralized</td>
</tr>
</tbody>
</table>

To view Slide 14, please see following page.
<table>
<thead>
<tr>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideas formulated</td>
</tr>
<tr>
<td>Release features defined</td>
</tr>
<tr>
<td>Design specs written/reviewed</td>
</tr>
<tr>
<td>Documentation development</td>
</tr>
<tr>
<td>System development:</td>
</tr>
<tr>
<td>Compiler</td>
</tr>
<tr>
<td>Supervisor</td>
</tr>
<tr>
<td>Host</td>
</tr>
<tr>
<td>Internal Doc. &amp; Testing</td>
</tr>
<tr>
<td>PROC conversion</td>
</tr>
<tr>
<td>Display Manager</td>
</tr>
<tr>
<td>Final documentation written</td>
</tr>
<tr>
<td>Testing (QA)</td>
</tr>
<tr>
<td>Education materials developed</td>
</tr>
<tr>
<td>Marketing</td>
</tr>
<tr>
<td>Distribution</td>
</tr>
</tbody>
</table>