Selecting an End User Programming Language for DSS Development

By: C. Lawrence Meador
Massachusetts Institute of Technology
Cambridge, MA

By: Richard A. Mezger
Entre' Computer Center
Boston, MA

Abstract

Today managers and policy makers are confronted with an overwhelming range of choices of computer software to develop decision support systems (DSS). The authors argue that DSS language evaluation and selection should be a multi-step process involving most, if not all, of the following:

1. End User Needs Assessment and Problem Diagnosis
2. Critical Success Factor Identification
3. Feature Analysis and Capability Review
4. Demonstration Prototype Development
5. External User Surveys
6. Benchmark and Simulation Tests
7. Programmer Productivity and End User Orientation Analysis

The objectives of each of these activities are described, as well as specific procedures for accomplishing them. In addition, the authors discuss the usefulness of a multi-disciplinary task force to accomplish the DSS language evaluation and selection process.

Keywords: Decision support systems, end user computing, software evaluation, architectural features, user needs assessment

ACM Categories: H.1.2, H.4.2, J.0, D.3.3

Decision support systems (DSS) represent a relatively new way of thinking about managerial use of computers. A decision support system is a computer-based information system that is designed to help managers in private corporations and policymakers in public sector organizations solve problems in relatively unstructured decision-making environments. Long-range and strategic planning, merger and acquisition analysis, policy formulation, policy evaluation, new product development, marketing mix planning, research and development, and portfolio management are a few areas where the DSS concept has been successfully applied [see 6, 9, 14 and 19].

Unstructured decision-making environments are those where the global problem is not well enough understood for a complete analytical description. A DSS is a system which provides computational and analytical support in situations where it is necessary to integrate judgement, experience, and insight of managerial or policy decision makers along with computer-supported modeling and presentation facilities. DSS focuses on achieving productivity improvements from managers and policymakers, rather than from the reduction of clerical and administrative costs.

As used in this article, a DSS language is one very important example of the generic set of computer application development tools generally known as end user programming languages (or in some cases, fourth generation languages). Other examples include relational database facilities with powerful report generation and ad hoc inquiry facilities; general purpose statistical data analysis languages; and broad based graphics generation languages. While each of these four types of languages usually address different user needs, they often share some subset of similar capabilities and characteristics such as:

- Integrated database management (sometimes relational)
- User friendliness to nontechnicians
- Both procedural and nonprocedural command structures
- Interactive on-line utilization
- Support of prototyping and adaptive development
- Modest training requirements for end users
- Easy debugging and intelligent default assumptions
Selecting an End User Language

— Quantity of code required only a fraction of Cobol, Fortran, etc.
— Internal documentation generation support
— Understandable code for non-developers [5].

The methodology presented here has been developed and used in software selection projects in a wide range of fourth generation software — including software for the emerging microcomputer marketplace. The DSS languages have been chosen as an example to illustrate the language selection methodology. This article deals with the process by which an organization acquires a DSS language (i.e., a tool to be used by end users and/or by analysts to develop DSS applications).

Introduction

Today’s managers and policymakers are confronted with an overwhelming range of choices of computer software to develop decision support systems for many of the important corporate applications referred to previously. Making the right choice of software for a particular organizational context can have a profound impact on the success of a DSS. A new, more powerful, cost-effective, productive, and flexible generation of software has now been developed and made commercially available for DSS applications. These so-called fourth generation languages are so much better than prior languages (such as Fortran, Cobol, Basic PL/I, etc.) that we believe DSS developers should only consider utilizing the prior generation of software tools in very unusual circumstances.

The purpose of this article is to address significant managerial issues in the evaluation and selection of a DSS language. Attention is focused on the critical areas of DSS end user characterization, problem diagnosis, and needs assessment along with their implications for the software evaluation and selection process. The role of top-level managers as well as data processing staff in the evaluation and selection process is also considered. (Some of these topics are also discussed in Meador and Mezger, “Decision Support Systems for Minis and Micros,” [13] and in Meador, Rosenfeld and Guyote, “Decision Support Planning and Analysis: The Problem of Getting Large-Scale DSS Started” [15].)

The selection and acquisition of a decision support language is too complex and important to exclude end users from the evaluation process. Top-level managers and their analytical support staff, who will be in the DSS user community, must participate in the evaluation and selection process in order to ensure that their needs are adequately addressed by the language selected. This discussion presents a range of methodologies and criteria which should be carefully considered by every organization embarking on a serious DSS development process.

A Multi-Step Process

The selection of an appropriate DSS language is an important and challenging undertaking. The necessity of matching the range of language capabilities to the range of organizational needs is crucial in light of the cost of computer and professional resources required to develop and effectively utilize the language. In addition, a formal process for DSS language evaluation and selection is an educational process. In the end the organization has a much better appreciation of what it needs, what it is buying, and the costs and benefits of accomplishing the planned improvement in managerial decision support. This education process not only improves the odds that the organization is making a good choice, it also provides a broad base of knowledge and appreciation of the facility being acquired and the process of deploying it. Thus, the educational component enhances the probability of successfully using DSS.

A multi-step process of DSS language evaluation and selection is needed. The individual steps within the process are shown in Table 1 and discussed in the remainder of this article. It may not be necessary to perform each step in order to complete an effective evaluation process. However, it is important to note that each step addresses different aspects of the planned use of the DSS language. Thus, each step provides additional knowledge that improves the chances of success (also see Keen, [5]).

268 MIS Quarterly/December 1984
Selecting an End User Language

Table 1
Steps in DSS Language Evaluation

- End user needs assessment and problem diagnosis (decision support analysis)
- Critical success factor identification
- Feature analysis and capability review
- Demonstration prototype development
- External user surveys
- Benchmark and simulation tests
- Programmer productivity and end user orientation analysis

Organizing for DSS language evaluation and selection

It is often useful to establish a multi-disciplinary task force to accomplish the DSS language evaluation and selection process. Our experience has shown that such a task force can accomplish its work in six to twelve weeks if the application domain is not extremely broad and if the number of people on the task force is kept small. We have observed such evaluation projects taking considerably more time in some organizations. We recommend that the group of individuals involved in the process include at least one senior manager and at least one representative of each of the major functional user areas which are designated to be DSS application areas (e.g., finance, marketing, research, sales, and so forth). There also needs to be representation from the data processing community so that all issues and consequences of hosting a responsive DSS environment can be adequately considered. In many cases, substantial computer systems resources are required for model utilization, data storage, and output display in a timely and responsive manner. Finally, the task force needs to include the individual(s) who will form the nucleus of the DSS support group (see Figure 1). This multi-disciplinary task force may be comprised of six to ten individuals who will play important roles in the evaluation and selection process (see also [2]). In most situations a core group of three, at most four, members of the task force should be charged with the primary responsibility for data gathering and analysis.

End user needs assessment and problem diagnosis

End user needs assessment and problem diagnosis is a systematic, organized, and structured procedure for identifying and evaluating features necessary for the DSS language. It involves direct contact with the intended users to understand the general nature of the business decisions which they are making and to identify the modeling, analytic, data manipulation, and display functions needed to support the business decision processes. Information is gathered by interviews and questionnaires; mechanisms that permit an experienced DSS analyst to understand the users' needs and to place these needs in the context of the facilities and features of DSS languages. (Research on DSS end user needs assessment is described in Meador, Guyote and Keen [11] and a structure for the process is recommended in [5].)

The end user needs assessment and problem diagnosis activity of the language evaluation process is similar to, but less detailed or intensive than the planning and analysis phase of the development life cycle. Both processes involve a DSS analyst exploring the nature of planned applications with its user community. For language selection, the objective is to determine the general nature and extent of the language functions and features which will be required to build the applications. Language functions and features which are not required are also identified. For actual DSS application development, the more detailed end user needs assessment and problem diagnosis is concerned with specific data structures, computational algorithms, and presentation contents and formats (reports and graphics).
End user education is an important component of the needs assessment and problem diagnosis process. In some situations, it may turn out that manual calculations of informal models are already in use to support the decision-making process—perhaps even with the use of alternative scenarios and “what-if” calculations. In such situations the educational process formalizes concepts and terminology to facilitate the dialogue. In other situations there may be little or no quantitative support of the decision-making process—manual or otherwise. It is necessary to introduce decision support system concepts and to explain to the manager how they might effectively be applied. It is important to keep in mind that the quality and quantity of the DSS support is expected to increase over time based upon the capabilities of the language being used and to the successful application of DSS concepts within the organizational unit and across the organization as a whole.

Thus, we see end user needs assessment and problem diagnosis as a means of improving the reliability of the statement of functional needs and requirements specification. This statement becomes the basis for the Feature Analysis and Capability Review stage. In addition, the interviews, questionnaires, and educational activities enhance the legitimacy of the feature specification and feature analysis part of the evaluation process. Finally, the process initiates communication between the decision-making managers and the technical specialists, a continuing two-way communication that is essential to success.

**Critical success factor identification**

After performing the assessment of user and organizational needs, it is important to prioritize the decision criteria to be addressed in the language selection process. This is done by identifying the factors which are critical for accomplishing the objectives of the DSS applications (and thus, by implication, the objectives of the key managers). These factors are referred to as Critical Success Factors (CSF), a concept introduced by John Rockart [17]. Critical Success Factors help to determine the key language features for which satisfactory performance will ensure overall project success.

Critical Success Factors for the DSS language selection process should reflect information gained in the interviews with users, executives, and technical personnel within the organization. By incorporating the input of personnel at all levels of DSS involvement, the CSF’s address user needs
in addition to economic factors such as budgets and projected growth, and technological factors such as system capacities and trends in hardware and software.

For each of the Critical Success Factors, a minimum level of performance is established and adequacy of each alternative language environment should be evaluated with respect to these criteria.

Feature analysis and capability review

General Features

The purpose of the DSS language feature analysis is to match the capabilities of the candidate DSS languages with the requirements determined in the user needs assessment and problem diagnosis activity. To do this, it is first necessary to select a number of candidate DSS languages for the evaluation. This process is a difficult one because of the large number of choices available. One of two situations is likely to exist. The first is that the host computer systems hardware has been selected — and in fact is installed and operational. In this situation it is necessary to determine the spectrum of DSS languages that are available to run on the selected computer systems hardware and within the operating system of that computer environment. In some cases, this process substantially narrows the choices of available DSS languages. The other situation is where the hardware selection is to be made upon the completion of the software selection. The latter situation presents substantially more language choices and also provides more latitude and flexibility for the language to quite closely meet the requirements of the organization.

In both cases there are a number of general considerations relative to language features which are important to the evaluation and selection process. These are listed in Table 2. These general considerations cover a broad range of areas of interest and are introduced briefly here. Compatibility deals with the manner in which the DSS language fits into the ongoing corporate computer systems environment, and whether it meets hardware, operating system, and data structure specifications. Availability issues are concerned with whether the language can be accessed on more than one hardware configuration and whether it can be used via remote computing services, and service bureau facilities as well as in-house. Frequently, the ability to begin DSS development out-of-house and then bring it in-house when usage increases and economics dictate can be very valuable. Maintainability considerations are concerned with how much effort is required to support ongoing use of the language. Reliability issues have to do with failure rates and recovery characteristics, and of the software's ability to perform all variations of its functions and to do so through the series of upgrades of the language which occur over time.

<table>
<thead>
<tr>
<th>Table 2. Language Features General Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatibility</td>
</tr>
<tr>
<td>Availability</td>
</tr>
<tr>
<td>Maintainability</td>
</tr>
<tr>
<td>Reliability</td>
</tr>
<tr>
<td>End user orientation</td>
</tr>
<tr>
<td>Programmer productivity</td>
</tr>
</tbody>
</table>

Two related topics are end user orientation and programmer productivity. They both involve the ease-of-use of the language and the amount of systems orientation, design, and programming skill required to use the language. Important issues here include whether the language has a procedural or non-procedural orientation. Some "languages" are essentially two-dimensional spreadsheet calculators with relatively restricted notations to refer to the rows and columns of the "model;" other DSS languages have comprehensive relational data management capabilities and substantially more powerful conventions and mechanisms for addressing data. Non-procedural languages with easy symbolic references to data and goal-oriented control conventions represent more of an end user orientation. Similarly, these features, representing more powerful and logically concise functions, are expected to improve programmer productivity during the initial development of a model and during the ensuing maintenance and modification cycle.

Specific Language Features

There are a number of technical considerations relative to evaluating DSS language features.
which are briefly summarized in Table 3 (a more detailed list is given in Appendix I). One of these deals with the design of the interface between the machine and the user, and the abilities of the language to provide a friendly and supportive environment for both novice and experienced users. In this regard, one needs to consciously distinguish between various “levels” of users as to how efficient they wish the system to be. (For a discussion of the user/system interface, see Sterling [18].)

Table 3
Language Features

<table>
<thead>
<tr>
<th>Technical considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface design</td>
</tr>
<tr>
<td>Data management — external, internal</td>
</tr>
<tr>
<td>Data analysis</td>
</tr>
<tr>
<td>Modeling</td>
</tr>
<tr>
<td>Data display — reports, graphics</td>
</tr>
<tr>
<td>Hardware/operating software environment</td>
</tr>
<tr>
<td>Multi-user interaction/sharing</td>
</tr>
<tr>
<td>Security and integrity protection</td>
</tr>
</tbody>
</table>

Multidimensional data management is often considered an important requirement of the DSS language. While there exits a substantial class of decision support problems involving only a two-dimensional analysis, it is more and more common to see three-dimensional and more than three-dimensional problems being modeled to support important organizational decisions. When the activities of a company are viewed by product line and geographic area, as well as by time and specific business variables (sales, depreciation, etc.), a four-dimensional problem exists. DSS languages approach the multidimensional manipulation of data in different ways, which is sometimes reflected in the level of complexity of the language code when developing a model to access the data. In other words, if multidimensional data models are to be constructed, the multidimensional data manipulation features of the DSS language should be powerful and easy to use.

Access to external data may also be an important requirement for a DSS language. External data for a model may come from other transaction processing or management information databases within the organization, on the same computer system, or on another one. Alternatively, it may be economic and demographic databases which are external to the organization and available on a subscription basis. The ability and ease with which a DSS facility can access these databases can be an important element in its successful use. The system-to-system interface to support this access needs to be “black boxed” because the day-to-day user of the application is often not a systems specialist.

Data analysis is the collective term applied to statistical and forecasting applications and requirements. There are numerous standard statistical and forecasting functions, many of which may need to be available for frequent use. Checklists for these statistical and forecasting functions generally exist in DSS literature or can be provided by the vendor of a DSS language. One should be modestly cautious here. User needs assessment activity may mistakenly identify requirements for more complex and sophisticated statistical and forecasting algorithms than are really needed. The issue here is to avoid selecting (or rejecting) a DSS language on the basis of the presence (or absence) of an advanced feature which, in reality, is likely never to be used. Alternatively, when such a sophisticated feature is required, but on an extremely infrequent basis, it may be appropriate to access that feature from a statistics language within a remote computing services (RCS) environment.

The modeling function is generally acknowledged to be the heart of a DSS application. Modeling is a process of representing, through mathematical equations and logical expressions, aspects of the organization’s business activities. By representing important parts of the organization and/or the competitive and external environments in which it operates, the model is able to support a careful study and analysis of alternative courses of action and outcomes. Such models often require complex sets of simultaneous equations where automatic equation reordering and simultaneous solution functions of a modeling facility are needed. “What-if” analysis evaluates the effect of changes in critical parameters. “Goalseeking”
provides a backwards calculation capability which, in essence, determines the parameters of the problem given the parameters of the solution. Hierarchical processing and consolidation across products, regions, and geographical entities frequently must be modeled. On occasion, multidimensional equations permit solution of highly complex models representing diverse and numerous parts of an organization. In some cases mathematical optimization routines (linear, nonlinear, integer, mixed integer, and goal programming, for instance) may be useful components of the language.

Data display of the results of analysis and modeling activity is, of course, essential. Results are usually printed in hardcopy or as video display outputs. Printed output is frequently needed in a rough format which permits the examination of intermediate values. When the analysis is complete, presentation-quality reports may be needed with appropriate formatting, labeling, and text. In graphics, features generally offered are line charts, scatter plots, bar charts (or histograms) and pie charts with options for multiple charts per page and various color, display, sizing, and labeling options (see [16] and [20]).

In a production environment, providing for access by more than one user at a time may be important. Options should exist to allow multiple users to execute the same model (program) to access different personal databases as well as for multiple users to access parts of a common historical database by the same or different programs at the same time. An important capability relating to simultaneous access is the ability to prevent simultaneous attempts to modify a database's contents. In general, a user community should have broad flexibility in terms of sharing models (programs) and databases [12].

Database security and integrity protection features are important to the successful use of a DSS language. It is common for many users to share the same computing system environment. Password control at the model and database levels is needed to protect the privacy of users, models, and data.

Vendor Support Considerations

In evaluating a DSS language, it is also necessary to consider the capabilities of the vendor who supports the language. Table 4 represents a number of vendor capability and support issues which need to be considered. All of the vendor capabilities shown are quite important; substantial deficiencies in any one of them could produce a painful and non-productive experience. Training and documentation are related capabilities which directly address the process of learning how to effectively use the DSS language. Training and documentation need to be available for a variety of levels and types of users so that first-time and infrequent users obtain adequate training in the basic of using the language while more sophisticated users are able to receive advanced training and have access to detailed reference documentation. In order to minimize the computer resources consumed, the efficient use of a DSS language becomes very important for large and/or multi-user models.

Two other related capabilities are crisis reaction and error/bug correction. There will be occasions, without doubt, when the application written in the DSS language does not work. The error diagnostics produced under such circumstances may be mysterious, or they may point to the DSS language itself as the source of the problem. In either case, there may indeed be a fault in the DSS language, or the model may have exceeded the capabilities of the language in some ill-defined manner. Crisis reaction or hot-line support means that the vendor has qualified systems personnel available on a standby basis to respond rapidly to such problems and to research and correct the difficulty. It may call for a suggestion as to how to program around the error or it may be a high priority activity to correct the problem and to provide updated software so that development and operation of the model can continue.

<table>
<thead>
<tr>
<th>Table 4 Vendor Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
</tr>
<tr>
<td>vendor facilities</td>
</tr>
<tr>
<td>in-house</td>
</tr>
<tr>
<td>Application systems develop</td>
</tr>
<tr>
<td>Consultation</td>
</tr>
<tr>
<td>Crisis reaction</td>
</tr>
<tr>
<td>Error/bug correction</td>
</tr>
<tr>
<td>Documentation</td>
</tr>
</tbody>
</table>
Finally, there are situations where it may be appropriate to look to the vendor for assistance in the design and programming of a DSS application. After all, the vendor is likely to be (and in fact should be) highly qualified and thoroughly experienced in designing and building applications using the language. Consultation involves relatively small amounts of support, more likely in the areas of design review or of designing and/or programming particularly difficult model segments. Application systems development support, unlike consultation, usually involves the full spectrum of development activities from design through deployment and is typically performed on a contractual basis. As to costs, application systems development support is almost always performed under contract and for a fee. Consultation may be for a fee or without fee — some vendors have limits below which the consultation is free and above which a fee is assessed. However, one should reasonably expect that brief periods of consultation via the hot-line support mechanism will be provided without fee. In this regard, there is typically an annual maintenance fee for ongoing support of a DSS language. In addition to providing updates to the DSS language and its documentation, the fee should include access to new language features and to training opportunities.

Acquisition Cost Considerations

One of the important features of a DSS language is its direct acquisition and utilization cost. Usually the cost factor which is given the most consideration is the initial acquisition cost. Important considerations when comparing the costs of DSS languages include the features of the language that are included, or not included, in the basic license cost. Some languages come in a totally bundled format while others are available in major functional segments (unbundled). Also, vendor support capabilities may or may not be bundled into the license fee; thus, there may or may not be a training allowance, a complement of technical documentation, and access to the hot-line. Even where there is an initial allowance, the cost evaluation should consider the costs of additional training, documentation, and start-up consultation support.

Other cost factors need to be considered. The cost of additional licenses for second and subsequent CPU systems are typically considerably less than for the first. The availability of the DSS language in a time-sharing or remote computing services environment, where it can be used and paid for on a fee-for-use basis, is often attractive. This is especially the case if models and databases can be moved from the remote environment to an in-house environment without costly conversion activities.

Another extremely important cost factor is that of the ongoing maintenance fee which frequently runs in the range of 1% of the license fee per month (e.g., a $75,000 license fee for a DSS language might well have a monthly maintenance cost of $750). The maintenance fee often guarantees receipt of the latest versions of the software and documentation. Further, it may very well provide for receipt of new and expanded features of the DSS language as they become available through the vendor’s continued program of support and enhancement.

Two related cost considerations are worthy of discussion. The first is the cost of the computer systems hardware required to operate the organization’s models when developed in the selected language. On the surface, the host computer environment may seem to be an almost limitless resource relative to the anticipated usage by the DSS community, although such a view is often deceiving. Experience shows that the use of a facility expands to meet the resources available. Models and databases grow in size, complexity, and frequency of use. In general, a DSS languages which will permit twenty busy users to obtain good throughput and response time in the host environment is substantially more desirable than one which only supports ten users with the same quality and range of performance capabilities.

The other cost consideration concerns the personnel costs needed to develop and maintain models written in the selected language. As will be seen, different levels of programmer productivity are possible when the same quality of individual uses different DSS languages. Consider that it might be substantially less expensive overall to invest funds in a more powerful host computer environment to permit the use of an inefficient DSS language (in terms of computer resources consumed) if the language is very efficient in programmer productivity.
Demonstration prototype development

Experience has shown that there can be significant value from using a language on a trial basis to develop a meaningful application before a decision is made to select the language for a major application or for corporate-wide deployment [1, 7, 8]. This approach involves the development and operation of a demonstration prototype. This prototype model should be based on a real need and should accomplish the major objectives of some planned DSS application.

There are several reasons why the development of a prototype application has significant value in the DSS language evaluation and selection process. It serves to:

— Verify the size and complexity of a meaningful DSS application as programmed in the target language.
— Quantify and measure the computer systems resources needed to develop and operate the application.
— Determine the characteristics and qualifications of the personnel needed to develop the application.
— Understand the training requirements for the use of the language.
— Test the vendors technical documentation and hot-line support necessary for effective model development and utilization.
— Determine estimates of programmer productivity with a given language.
— Demonstrate the feasibility of the DSS application and obtain operational experience in the application area.
— Experience the entire DSS application development process from an educational perspective in what is clearly a realistic setting for future uses of the DSS language.
— Obtain at least some real decision support value in a limited timeframe, i.e., actually build, deploy, and use a DSS application which has immediate value to corporate management.

It is often true that in-house professional staff will be used to design, develop, and deploy the prototype DSS application with support from the vendor. It is possible that support from outside consultants specializing in DSS software evaluation, selection, and application implementation would provide assistance on a cost effective basis in the prototype development as well.

External user survey

An external user survey is an evaluation of the experience of other organizations using the DSS languages being considered. The user survey is of particular value in an important business undertaking such as DSS development since it provides insights into key considerations and potential problems. Specifically, the objectives of conducting an external user survey are to:

— Obtain independent and unbiased information on the performance of the DSS language and of the vendor.
— Identify potential problems as well as sources of strength and weakness of candidate DSS languages and of their operation in particular host hardware environments.
— Verify computer systems hardware and support software requirements across a range of DSS applications.
— Develop realistic implementation planning information including an understanding of training requirements and implementation productivity considerations.
— Assess the end user managerial and technical staff satisfaction with the candidate DSS language.

This process of performing an intensive reference check can be done in stages as the DSS languages move closer to final selection and as some candidates are eliminated. Initially, an informal, but organized, one-half to one hour telephone conversation with the DSS coordinator of an organization using the candidate DSS language will suffice to obtain a useful impression. As the finalists emerge from the selection process, more in-depth interaction with heavy users of the language, including an on-site visit, can be quite valuable and is definitely encouraged. The vendor can be helpful in providing names of candidate organizations to choose from, as can be a
A major objective of the external user survey is to determine overall satisfaction with the DSS language. Dimensions of “satisfaction” include ease of learning and use, quality of documentation, programmer productivity, and efficiency of performance. Table 5 presents an overview of the information which is sought in the external user survey process.

Vendor performance is to be examined in two areas: 1) the timing and smoothness of the initial installation and any modifications and upgrades subsequently provided; and 2) the overall quality and quantity of ongoing vendor assistance. Aspects of vendor assistance to be examined include: initial installation, user support, product maintenance and upgrades, technical competence in providing support, reliability of product, cooperativeness and availability of support, and availability and timing of training. The analysis of vendor assistance should be summarized by determining and understanding both the strongest and the weakest points, as the organization will seek to maximize the vendor’s strength and to minimize or bypass vendor weaknesses.

Experience has shown that many corporate users of DSS languages are quite receptive to participating in such an extensive reference check. They have an opportunity to demonstrate their success as well as to discuss potential DSS applications and how the applications provide decision support assistance to their management. A well-organized approach with specific information-gathering objectives is likely to be most successful in developing a receptive and cooperative relationship and in obtaining the desired information.

**Benchmark and simulation tests**

At first glance it might appear that benchmarks and simulation tests are similar to, or even redundant with, a demonstration prototype project. While there is a general similarity in the objec-
Selecting an End User Language

tives, the two activities are quite different in an important and meaningful way. A benchmark is a series of simulated tests of a comprehensive set of the features of the DSS language. It seeks to determine the level of computer systems resources utilized by the various capabilities of the DSS language (see Table 6). With these objectives, the programs or models which comprise the benchmark do not, in general, solve real DSS applications problems; rather, they are specially constructed to exercise various features or capabilities of the DSS language in a known manner.

Table 6
Benchmark Evaluation

| Measure Computer Systems Resources Consumed by DSS Software in Typical User Operations |
|---------------------------------|--------------------------------------------------|
| • CPU Cycles                    | • Main Memory — including virtual memory paging load |
| • Large-Volume Disk Input/Output | • Input/Output Activity                          |
| • Response Time                 |                                                  |

For example, a benchmark program may seek to shed light on the amount of computer main memory consumed by a typical model and the way in which the memory is managed by the support software. This benchmark program is constructed to consume a predetermined amount of memory, although the way in which it uses the memory only approximates the manner in which a real model uses memory. All of this contrasts with the demonstration prototype approach where the emphasis on solving a real DSS application problem may very well require only a small subset of the total DSS language features and capabilities to be exercised.

The objectives of a benchmark evaluation are to:

1. Measure computer systems resources consumed by the DSS language in typical user operation (see Table 6 for resources to be measured).
2. Determine cost of computer resources consumed (where costs are understood to be determined to be a usage algorithm intended to simulate the utilization of specific computer resources).
3. Learn about programming with the DSS language.
4. Check and verify the operation of a number of important language features and capabilities.
5. Evaluate the user friendly or English-like features of the DSS language.
6. Improve confidence in the DSS language’s features and capabilities to meet the needs of the organization’s DSS applications.
7. Improve confidence in feasibility of DSS application development.
8. Develop a series of sample programs for analysis of programmer productivity and user orientation.

In essence, the benchmark activity enhances the user needs assessment activity by forcing the analyst and the user to think through in more detail the specifics of the needs. It permits exploration of important systems features and often exposes multiple ways of meeting a user requirement. It also requires the use of the DSS language’s technical documentation and provides an opportunity to test the vendor’s technical support. Finally, when the benchmark is run on an in-house computer, it allows for an observation of the installation process, and, quite likely provides experience in moving DSS models and their data from one computer system environment to another. In summary, Table 7 presents one possible set of benchmark components for consideration.

**Programmer productivity and end user orientation analysis**

Two additional issues of importance in the selection of computer software for DSS development, maintenance, and enhancement include the extent of programming development effort that must be expended to achieve a given set of objectives (a function of programming language productivity), and the range of types of individuals who directly use the language who may reasonably ac-
Selecting an End User Language

Table 7
Benchmark Components

<table>
<thead>
<tr>
<th>Databases: Statistical and Financial</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Benchmarks:</strong></td>
</tr>
<tr>
<td>• Data entry</td>
</tr>
<tr>
<td>• Financial model calculations</td>
</tr>
<tr>
<td>• Goalseeking (backwards iteration)</td>
</tr>
<tr>
<td>• Aggregation</td>
</tr>
<tr>
<td>• Data communications</td>
</tr>
<tr>
<td>• Large-model startup</td>
</tr>
<tr>
<td>• Linear regression</td>
</tr>
<tr>
<td>• Basic descriptive statistics</td>
</tr>
<tr>
<td>• T Test</td>
</tr>
<tr>
<td>• Matrix correlation</td>
</tr>
<tr>
<td>• Exponential smoothing and moving averages</td>
</tr>
<tr>
<td>• Growth rates</td>
</tr>
<tr>
<td>• Volume data transfer</td>
</tr>
<tr>
<td>• Report output</td>
</tr>
<tr>
<td>• User interface</td>
</tr>
</tbody>
</table>

In general, it can be stated that “more productive languages” support the achievement of end user application goals with less total program development effort (and thus cost) than would be expected of less productive languages. More end user oriented languages are accessible by a wider range of users (because the languages are more like the “natural” language of the users and thus are more “user friendly” [4]).

More productive languages tend to require less specification of the detailed procedures by which desired goals are achieved. These highly productive languages are often referred to as goal-oriented or non-procedural languages for this reason. They tend to require the specification of fewer lines of executable code and fewer lexical items (shorter and simpler lines of code) to accomplish a given purpose. User-oriented languages tend to emphasize logical names of entities such as variables, commands, labels, locations, logic, and dimensions, rather than numeric codes or highly constrained acronyms.

Some fundamental criteria related to productivity and end user orientation which can be applied to DSS applications are:

- Executable lines of source code in an application program — excluding program comments.
- Lexical entities in the programs — excluding program comments, where a lexical entity is any continuous character string of one or more characters with meaningful definition such as a variable name, command, label, data item, logical or arithmetic operator, etc.
- Average number of lexical entities per line of code in each program.
- Numeric string ratio in each program — defined as the total number of numeric strings divided by the total number of lexical entities.

Programmer productivity analysis is an important part of the language evaluation process that can reveal potentially large hidden costs in language acquisition. Dunsmore and Gannon [3], for ex-
Selecting an End User Language

ample, show that significantly different levels of programming effort can exist between almost-identical languages. However, it should also be noted that much disagreement exists among both academics and practitioners on the proper metrics for measuring programmer productivity and end user orientation characteristics of different languages.

Multicriteria Assessment

It would seem that much of the language evaluation process could be facilitated by some sort of multicriteria scoring or weighting scheme. We have observed, and sometimes used, schemes which assigned weights and point scores to different functions of the language and to outcomes of other aspects of the evaluation process. The merit of such an approach is that it collapses results and evaluations among several dimensionally incompatible criteria into a single metric, and produces a simple scalar comparison to rank candidate languages. But in presenting recommendations to user management, we think that relevant summarized raw data in its native dimensions should still be presented. User managers may have differing weights which will change over the evaluation timeframes as they learn more and more about the issues that count.

Conclusion

The selection of an appropriate DSS language is a challenging and important task for organizations that are starting to focus on information technologies to improve the effectiveness and productivity of managers and policymakers.

Both the process and the structure of the language evaluation activity are likely to impact its effectiveness and its success. The software technology for decision support is changing rapidly and substantial variance exists in the quality and relevance of the hundreds of products designed for potential decision support applications. Investment in a careful, well thought out and credible user-driven evaluation process is likely to be worthwhile but care should be taken to avoid studying the alternatives so long that the decision support opportunity evaporates.

References


About the Authors

Richard A. Mezger is the chief executive officer of an Entre’ Computer Center in Boston, Massachusetts and has been actively involved for more than twenty years in the information systems field as a developer, consultant, and teacher. He received his Master of Science in management from the Sloan School of Management, MIT. His consulting experience and publications have focused in recent years on evaluation and acquisition of decision support software for both microcomputers and mainframe systems as well as development of large scale institutional decision support systems. His current activities involving the retail sale of personal computing systems and services have a major DSS thrust.

C. Lawrence Meador is a Lecturer in the School of Engineering, MIT, and is President of Decision Support Technology, Inc., a consulting firm in Waltham, Massachusetts. He received his graduate degrees from the Sloan School of Management and the School of Engineering, MIT. He directs the Decision Support Working Group of Nolan, Norton & Company and he serves on the board of directors of Access Data Services, Inc. His current research is concerned with future DSS technologies and impacts, end user needs assessment, and software technology development and evaluation. He has lead the development of several large-scale multi-user DSS. He serves as an editor of Computer Communications and Communicacion e Informatica.
Appendix 1
SAMPLE CRITERIA FOR EVALUATING DSS TOOLS

A. FUNCTIONS AND FEATURES
1. Modeling — able to calculate with the information in the system, do optimization, “what-if” analysis
2. Procedurability — ability to solve equations independent of their ordering, symbolic reference of data
3. Data Management — number of dimensions, handling of sparse data, ad hoc inquiry
4. Report Generator — ability to produce high quality formal reports quickly and easily
5. Graphics — line, pie, bar, quality of output
7. Project Management — PERT/CPM, multi-level work breakdown structure
8. Operations Research — linear, integer, dynamic programming
9. Forecasting & Econometrics — time series analysis, seasonalization, smoothing
10. External Databases & Interfaces
11. Security — database, file, model, class of user

B. EASE OF USE
1. End User — analysis performed directly by person who needs the information
2. Programmer/Analyst — interested in the quality of the editor, data management, report writer, etc.
3. Ad Hoc Inquiry — end user answering questions for which no standard report is available

C. FACILITIES
1. Documentation — for user, programmer, operations
2. Training — novice/advanced, systems/user
3. Support — consultant, hot line
4. Host Hardware — computers supported
5. Operating Environment — operating systems, disk requirements, etc.
6. Availability — in-House & on Timeshare

D. MARKET POSTURE
1. Pricing — lease, rent, purchase
2. Installations — number of users, length of use
3. Target Market — type of business actively pursued by the vendor
4. Plans — commitment to DSS as a business area, amount of R&D
5. User Perceptions — degree of use and support, functions used
6. Vendor Viability — size of company, revenues, etc.