Application generators at IBM

by AARON M. GOODMAN

IBM Corporation
San Francisco, California

ABSTRACT

This paper discusses the reasons for the great interest in application generation. Because of the growing backlog of user demands on data processing, a new technology with an order-of-magnitude increase in productivity is suggested. IBM has two families of application generation products. One, DMS, operates in a CICS/VS environment; the other, IMSADF, operates in an IMS/VS environment. An overview of the techniques involved and the benefits of using these systems is discussed.
The most important problem facing data processing management today is the backlog of applications. There are a number of studies done by IBM and other groups which show the backlog to be many years long. Figure 1 shows the results of some studies on application backlog. The trend for the future indicates an even greater need for data processing solutions. The labor force is growing more dependent on information which must be provided to do their jobs. We, as individuals, demand more and more information in a timely fashion; it would be inconceivable, for instance, to go back to the days where airline reservations and ticketing were not immediate. As consumers, we ask for more and better service.

**Application Backlog Growth**

<table>
<thead>
<tr>
<th>Year</th>
<th>Have Application Backlog (%)</th>
<th># of Applications in Backlog</th>
<th># to be Operational by End of Year</th>
<th># of Online Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1977</td>
<td>94%</td>
<td>5</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>1978</td>
<td>96%</td>
<td>15</td>
<td>3-4</td>
<td>3</td>
</tr>
<tr>
<td>1979</td>
<td>97%</td>
<td>20</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 1—Statistics on application backlog

One reason for the large and growing backlog is, as mentioned, the increase in service that we demand from data processing groups. There is another important reason, i.e., the availability of skilled data processing professionals. Figure 2 shows the expenditures for programming, as reported by Diebold, across industry. We see that 24% of the data processing budget is for programming, but 69% of that 24% is used to maintain existing systems. Therefore, only 8% is available to develop new applications. The backlog of applications can be addressed in only a small way, since the bulk of the programming resource must be used to maintain current systems.

The difficulty of finding and attracting qualified programmers is well known. Figure 3 shows some results of studies and indicates that the future is bright with prospects in the programming field. Put another way, it's going to be more difficult and much more costly to satisfy our data processing needs by simply hiring more programmers. This is already evident when we recognize the decline in the percentage of DP budget due to hardware costs and the increase due to software costs. Due to the educational philosophies of various countries, this problem may be more or less important, depending on geography.

So, we find a situation where more data processing solutions are required, but with severe constraints of people and budget to get the necessary systems.

**The DP Dollar**

- **New Applications**: 8%
- **Maintenance**: 16%
- **Programming Personnel**: 24%

Figure 2—Programming expenditures

In addition, the advent of major DB/DC systems such as IMS and CICS brought two long-existing problems into sharp focus. One is data security, a topic of much discussion in data processing. Now it is a more significant topic since more data are located in a single or a few repositories. The data, being more centralized and more complete, are more valuable. The results of destruction or unauthorized usage are more profound. In addition, sharing of the data processing resources, database, and programs has caused more attention to the topic of change isolation. How to share resources, while still retaining independence for individual systems and being able to change systems without widespread effects on other sys-

**Programmer Availability**

<table>
<thead>
<tr>
<th>Year</th>
<th>Need: 640K</th>
<th>Supply: 476K</th>
<th>26%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1974/75</td>
<td>Need: 320K</td>
<td>Supply: 306K</td>
<td>4%</td>
</tr>
</tbody>
</table>


Figure 3—Statistics on programmer supply and demand

From the collection of the Computer History Museum (www.computerhistory.org)
tems, is a significant problem which is accentuated by DB/DC systems.

Many of these problems can begin to be addressed today by using a new tool—Application Generators. This "new" tool is, in fact, a continuation of a trend we've had in data processing since the 1950's. Remember machine language? We've come a long way since then, and Figure 4 will allow us to reminisce a bit.

Evolving Programming Technology

IBM has developed two families of Application Generator products in the large system environment. One is based on usage of IMS/VSE; the other is based on CICS/VSE. In the early 1970's, people in the Rochester, Minnesota, manufacturing plant of IBM developed a system to assist in solving a number of problems with application delivery at that plant. That was the granddaddy of IMSADF. At about the same time, people on the East Coast were developing a system to assist the development of online systems (particularly screen design and development), and this was the start of DMS/CICS/VSE.

We now have a number of years of experience with these systems and have worked with customers to support more and more requirements. From an architectural point of view, a generator is feasible because it makes it possible to identify and separate out functions which are common to many applications, code them separately, and put them together in a way that supports a particular application need. The programmer, then, needs to work with specifications which relate to the business needs to be addressed, not primarily to the programming required.

Let us look at the functions in an online program. Architecturally, most business data processing applications will fit into the framework of Figure 5. We have, in addition, coded the logic of each of these functions in a generalized manner and have developed a technique to supply specifications through a source external to the common program modules. Therefore, we have decoupled the FUNCTIONS which satisfy the application requirement from the PARTICULARS of that exact transaction.

The usefulness of a particular generator depends, in large measure, on its richness of function. It is clear, I think, that we'll not get to the point where all application requirements are satisfied with common code alone. Both ADF and DMS allow for EXITS and Special Processing, where the programmer gains control and programs in a traditional manner. Where this is required, the increased productivity promised by generators is reduced. Therefore, the intent is to, over time, put more and more of these functions into common code.

The benefits of this technique accrue throughout the development cycle—during design, during programming, in test, and during maintenance. During design, the ability to work closely with a user and prototype the solution quickly, enables the user and the programmer to understand the needs and agree on the system. The wide gulf between user and the DP function is often crossed by simply showing a proposed solution, or at least an approach.

During programming, much of the work is reduced because much of the code is supplied. The programmer often works at the external level of the system, specifying the particular requirements for this particular application. If security is to be enforced, for example, the programmer now supplies DATA about the authorization, rather than reinventing another security module. Even if some code must be written because all the needs are not addressed, it is much less than if the whole application were written conventionally.

Figure 4—Trends in programming technology

Traditional Application Development

The approach to testing can be quite different if a generator is used. The common modules of code are pre-tested, and if problems occur, they are to be corrected by the vendor. But, even more important, hundreds of customers are running these exact common modules; therefore, they tend to be thoroughly debugged. Testing of a particular application involves, by and large, verification of the parameters and business information that the programmer provides to the generator.

All of the above contribute to the improved maintainability of generator-produced solutions. But in addition, a person can easily maintain their application because changes are usually external, and the common code is left alone. It is also much easier to maintain someone else's application, since the need to understand the data processing techniques used is lessened. Parameters, constants and business rules change, and these in fact are held separate from the common modules.

Because of these benefits, we can now begin to approach the growing backlog with a tool which offers an order of magnitude of improvement on our productivity.