SOFTWARE DEVELOPMENT IN TRANSITION:

USING TRANSACT / 3000

by Norm Wright

Summary - The paper presents a comparative study of programmer productivity on several recent development efforts on the HP3000. The development projects are good subjects for comparison, since the software effort was of a similar size and scope, and each used a dedicated staff with approximately equal capabilities. They also involved the same customer and demanded that the end product be comparable from the point of view of the user. Two of the projects involved Cobol, Image, and VPlus; the third used Rapid/3000, including Transact, almost exclusively. Statistics are presented which delineate the modest productivity savings which were achieved. Additional improvements which were achieved in user presentation are mentioned, along with a brief discussion of the expected impact of Transact on program maintenance. Additional discussion will focus on the the experiences of the development staff in adapting to Transact/3000. Changes in concepts, procedures, and strategies of Cobol programmers using the new dictionary-based development methodologies are discussed.

For the past three years, our organization has been involved in a number of medium to large-scale software development efforts, all on Hewlett Packard 3000 equipment. Among three recent projects, two involved the conventional HP3000 development triad: Cobol, Image, VPlus. A third major project launched boldly into the fourth generation, utilizing HP's most recent software development tools: Dictionary/3000 and Transact, with its built-in access bridge to VPlus and Image. The discussion which follows is based upon perceptions which evolved from this succession of projects.

For most users of HP3000 equipment, there are several initial deterrents to undertaking major software development effort in Rapid/3000. One deterrent may be the high performance overheads which are reportedly experienced in production software using Transact. Almost all users who have gotten beyond casual or experimental use of Transact sooner or later will experience this problem -- this is true at least with the present generation of HP equipment. Fortunately for those users, the well-known problem has an equally well-know solution. The solution comes in the form of a well-known Transact optimizing compiler which has become a standard remedy for users who are experiencing performance headaches.

A second major deterrent may be the high cost of the Rapid/3000 package. For many small organizations, especially those in competitive markets, the cost of development software itself may seem prohibitive. Only if the user organization is large and spends a considerable amount on in-house software development is the cost of Dictionary/3000 likely to be justifiable. If the differential can reasonably be expected to be absorbed by the reduced development or maintenance costs of new systems, the software becomes cost justifiable.

Unfortunately, the subject of developmental time and cost savings has seldom been reasonably addressed. Vendor claims for the economies which can be expected from the implementation of fourth generation methodologies tend to be moderately to wildly exaggerated. Claims ranging from 100 to 1000 per cent leave the prospective user of fourth generation languages feeling like the incredible shrinking man. Above all, such vendor claims seem rarely based upon realistic and reasonable assessments of practical user situations. The discussion which follows is an attempt to realistically measure the impact of Transact on a real life software development process.

Table 1 presents the historical experience of our organization in software development with three recent projects. The columns C1 and C2 both represent Cobol-Image-VPlus software development, while the column T1 represents the development effort which utilized Transact almost exclusively. The three projects offer a variety of built-in advantages for comparison. All were carried out with a similar staff makeup, comprising a range of programming skills and experience levels. Unique to the working environment, all three development efforts were carried on with "captive" staff -- the personnel were dedicated completely to the project at hand and had no other professional duties which were outside the area. All three projects were designed and implemented for the same client, hence all three attempted to hit a level of sophistication, user expectation, and documentation that was approximately the same.

The development time figures are expressed in man months, which have been adjusted for overtime and holidays to reflect a work week of approximately 40 hours. The figures shown represent the total development, beginning with the systems analysis and database design. The time overheads involved in loading the data dictionary are included for the Transact code, as well as the time for building VPlus forms in both Cobol and Transact. Program design, coding, and debugging times are included, as well as the time necessary to integrate and test the system prior to implementation. The man-month figures also include the time involved in producing reasonably complete user, program maintenance, and operations documentation for each system. They do not include the initial project definition and functional specification phases.

Measures of programmer productivity which have been applied in the past frequently relied upon the total number of lines of code produced. Accordingly, the table presents the total number of lines of code comprising each system, broken down for comparison of interactive versus batch programs. Seperate categories show the size of associated copy libraries, include modules, and other supporting code.

		System C1	System C2	System T1
1.	Interactive Programs	34	37	64
2.	# Lines Code, Interactive	26,720	37,714	60,529
3.	Average Prog Size Interactive	786 lines	1019 lines	946 lines
4.	Batch Programs	58	35	56
5	# Lines Code, Batch	40,047	28,022	29,046/48 *
6.	Average Prog Size Batch	690 lines	801 lines	605 lines
7.	Total Programs	92	72	120
8.	# Lines Misc Code: Copylib/Include/etc.	4,224	4,388	3,758
9.	Total Lines	70,991	70,124	93,333/112 *
10.	Average Prog Size	772 lines	974 lines	833 lines
11.	Total Man Months	74	52.5	60
12.	Man Months per Program Module	.80	.73	.50

* Eight batch COBOL reporting programs in system T1 averaged 904 lines each.

- TABLE 1 -

SOFTWARE DEVELOPMENT STATISTICS

FROM THREE SYSTEMS DEVELOPMENT PROJECTS

No adjustment is made for the Transact system to attempt to capture the size and coding involved in the data dictionary itself. The figures for the Transact system also do not include a significant (and growing) number of Report and Inform auxilliary programs.

In comparing Transact with Cobol, one would intuitively feel that a measure involving the total absolute number of lines of code would be inadequate. After all, one of the most well-publicized features of fourth generation languages is their syntactic brevity, and the ability to condense complex programming structures into a short sequence of statements. But this reputation for brevity is not borne out by the absolute number of lines of code produced in these three projects. Instead, Table 1 shows as much variability between the two Cobol systems in terms of the number of lines of code per program, as it does between the Cobol and Transact systems. Of course, had the statistics extended to the level of capturing the number of words or characters produced, then it is very possible that the fourth generation's good reputation could have been restored in this respect.

But even though the figures do not show a consistent differential between Transact and Cobol in the number of lines of code produced, it seems wise not to base any measure of productivity upon such a measure. A better measure of productivity would make some attempt to quantify the program complexity. In particular, it would seem relevant to try to measure the functionality of the programs produced -- the overall "work" or "complexity" of the resulting system. Ideally one would like to know how much time would be saved in writing exactly identical systems -- one using each development methodology.

Fortunately, in the case at hand, there is no need to attempt to develop a new programmer or productivity measurement. There is a much easier way, offered by the fact that all three systems were produced at a similar level of design sophistication. While the three systems are not identical -- in fact they are quite different applications -- they do represent perhaps the closest it is possible to come to this condition in a real world setting.

Since all three systems are similar from the standpoint of both design and user presentation, we can take the program itself as a functional unit of measurement. This is to say that, roughly speaking, given the three systems presented here, a single program from any system behaves in the same way and performs the same functional "work". Of course some programs will be more complex, accessing a large number of datasets and performing many functions. But overall in the three systems shown here the total "functionality" of the system is roughly proportional to the number of programs in the system. Such a measure would, of course, be inappropriate for systems which were designed to different standards of either functionality or modularity. But the three systems under consideration offer somewhat of an ideal case in this regard.

Using the program itself as a measure of functionality, the productivity measure which is listed last in the attached table can be derived, the figure "Man Months per Program". This measure can be used to give a close approximation of the overall time savings, experienced in these projects, of Transact over Cobol. As the table shows, programmers on the Cobol projects required between .7 and .8 man months to produce each functional program module. The time for a similar module using Transact was .5 man months. This would suggest that productivity improvements in the range of 28% to 38% were experienced.

The reader will note that the variability between the two Cobol projects (C1 and C2) is one third as high as between Cobol and Transact. In fact, the project C1 involved high startup costs in terms of a team of programmers getting oriented toward a new client and a new environment. The time savings which were experienced in C2 are more the result of a team pursuing established methods, and capitalizing on working tools (program shells, drivers, database routines, copy libraries, etc.) established during the first project. To a certain extent, some of these established methods were also useful in T1 -- the Transact project -- in that they did at least provide an existing standard which had only to be adapted to the new programming methodology. Although the startup overheads of learning and adapting to Transact were considerable, especially in the first few months of the project (T1), the project was also facilitated in many phases by being the third in an established series. Overall, it is at least convenient to assume that the overhead of learning Transact in T1 was counterbalanced by the beneficial effects of the pre-existing models in Cobol.

An additional factor is not satisfactorily reflected in the statistical presentation in the table. The use of Transact in T1 facilitated a number of significant improvements, particularly in the area of user friendliness and ease of use. As two examples, the increased use of sorted record presentations and the increased capabilities of browse-type record accesses in T1 could be cited. Both types of presentation are vastly easier in Transact than in Cobol. The programmatic ease of such capabilities in Transact meant that many functions in the T1 system utilized the features, where the nearest Cobol equivalent did not. Overall this did lead to a significant improvment in user presentation for the T1 system users over that provided by the two Cobol systems. Our installation does not appear to be unique in this experience. The increased sophistication of Transact code from the user point of view has been mentioned by a number of other users of Transact.

Another very interesting topic which we will not attempt to cover here is the matter of improvements in program maintenance to be expected from Rapid/3000. To some extent, program maintenance is a continuation of the same processes which were going on in development. Programs are being enhanced and extended, screens are being changed and rearranged, and the database must also occasionally be revised and restructured. This would suggest that as a minimum, we could expect the same economies as during the development process.

Our own preliminary experiences with maintenance of a Rapid/3000 system would suggest that this is true. In addition, it would be wrong not to add that the capabilities of Report and Inform have also proven to be tremendous advantages. The defacto and specialised reporting capabilities are especially useful in reducing the backlog of maintenance activities. This allows the maintenance staff to be in a better position to react to the more complex and far-reaching requirements of users. This is the reverse of the maintenance predicament at most installations, where the steady background of minor changes and request overwhelms the staff resources long before more complex changes can even be considered.

Well, the productivity gains which were experienced during the development phase of our project were modest, especially in view of the elaborate claims which have been made for some fourth generation languages. Perhaps it is important at this point not to forget that productivity gains in excess of 25% in almost any industry are bound to have important, far reaching impacts. Above all, it is probably wrong to expect the entire process of computer analysis and software design to vanish into nothing at this particular stage of its development. Whether it is the fourth generation or natural language interfaces, important problems still remain in the formulation of complex, real-world problems into computer algorithms and processes.

Norm Wright - has worked on a variety of software systems for HP3000 computers since 1975. He has spoken at previous Interex conferences in Baltimore, Orlando, and Copenhagen. He is currently associated with INFORMICA, an independent software firm, doing business in the Kingdom of Saudi Arabia.