Management Options For The 80's

Giles Rider
Project Manager Rex Development

Mr. Rider has been reponsible for development of PAL — the User-friendly Front-end to REX/3000, Gentry, Inc.'s general purpose programming language and report writer. Gentry, Inc. is a HP OEM and Contracting company in Oakland, Ca. Mr. Rider was educated at MIT and the University of Connecticut. Prior to joining Gentry, Mr. Rider was an independent consultant.

INTRODUCTION

HP's choice of the phrase "Interactive Information Management" as it's theme for the beginning of the 1980's reflects the changing emphasis and the quickening pace of the industry as a whole. By the middle of the decade much of today's hardware and software will be obsolete, and many traditional DP concepts will also have fallen by the wayside. The continuing rush of hardware developments — increasing speed and reducing cost per calculation — make it practical to cost-justify new applications daily. New powerful software — Data Dictionaries, report Generators, turnkey applications — holds out a lot of promise for simplifying application development. Yet each new application presents risks - both personal ones, in terms of job security, and company risks, in terms of survival and growth of the business.

In this paper we seek to identify guidelines and principles that can be followed to minimize these risks — guidelines that will also help to identify new opportunities as they arise, and to exploit them fully. We will examine each of the areas that are mentioned in the title and attempt to provide ideas for action that are both practical and useful. These ideas are intended to supplement, rather than to supplant the conventional wisdom on each of the following subjects, while providing a unified viewpoint — a grand scheme — from which it all makes sense.

Before we begin the detailed analysis of the areas, a few general remarks are in order. Economics indicates that the scarcity of a resource determines its price, and that cheaper resources are always substituted for more expensive ones when they are available. When the CPU was the most expensive component of a system and software was cheap, it made sense to use keypunches and run all jobs in batch mode; now it doesn't. Now that maintenance of old applications is the single largest people cost in DP, does it still make sense to keep building applications that will need to be maintained in the traditional ways? We saw that CPU power was substituted for people power in the transition from batch to on-line; doesn't it make sense economically now to begin to substitute CPU power for people power in the area of system maintenance?

Applications development is another area where people cost is the major expense — and this isn't because application development is the kind of "creative" activity where only "artists" can do the work. More often than not, the main reason for the cost is that suitable automated tools either do not exist or are not used, so that manual reports are the mainstay of most development projects. Many automated project control systems stress accounting for the cost of the project more than supporting the design and implementation activities, which reflects DP's traditional accounting-support function and orientation.

The use of Data Dictionaries in applications development represents another application of CPU power to reduce the effort involved in application development — and in maintenance as well. In application development, the use of dictionaries tends to bring disagreements about the use and meaning of the data out early enough in the project so that the conflicts can be resolved in the system design, rather than by after-the-fact administrative procedures and manual forms. In the maintenance area, where about 70% of the programming dollars are currently spent, a properly constructed dictionary can result in enormous savings, as the dictionary is the only place changes to applications have to be made by hand, and proper safeguards can be set up and followed to ensure that the integrity of the databases is not compromised.

Application packages can also help to ensure that the limited DP budget buys the most in DP service. Application packages cost so much less than "home-made" programs that the justification of "in-house" programming for common applications like A/R is very difficult, especially for new users. Another powerful reason for moving to packages is that the shortage of programmers makes it disadvantageous to waste them on re-inventing a General Ledger system, when they could be creating a new system for your company that could give it a lead on its competition. Programmers know this too, and are more likely to stay with companies that promise them a future working on new development and "state-of-the-art" projects than with companies that offer maintenance of an obsolete G/L system.

Contract programmers are about the only ones who are willing to get involved with maintenance programming as a steady diet, and if you use them properly, they can benefit your company, allowing you to make more profitable use of your regular employees, while not causing salary riots because of their higher rates, because they are doing the work that no one else wants to do.

Microcomputers in user departments are often seen as a symptom of DP department unresponsiveness — a "sagebrush rebellion" in the user area. There's an old saying that a problem is just the wrong way of looking at an opportunity, and that's especially true of the Microcomputer. A proper response to the Micro invasion can actually improve the situation, both for the DP department, and for the users; cutting costs while building user confidence in your DP expertise.

Programmer productivity tools, software tools, Programmer's workbench — all of these buzzwords carry thru the concerns that we've previously mentioned — how to apply CPU power to make the programmer's job easier. But there's two sides to this one, and there's good reason for caution. Here's where the approach that de-skills the programming function can cause the same kind of worker problems that plague the assembly line factory and the so-called "office of the future." Yet good productivity tools do exist, and should be used whenever possible to reduce the drudgery and repetition that is often found in programming. Programmers are one of your most expensive resources, and to watch one using a line editor can be a real eye-opener.

DATA DICTIONARIES:

Rule #0: Select the right data dictionary for your environment.

Data Dictionaries differ in capabilities and in their interfaces to the outside world. Ideally, you want a Dictionary that allows you to create your application code via the most powerful application generator that your environment will allow (this could be a manual application generator — i.e. a programmer — or an automated one that produces COBOL or other source code.)

Your dictionary will also be a central place for manual changes — when the length of a data item changes, for instance, you should only have to change the entry in the data dictionary, not in the programs that access the data. You'll see the importance of this when you remember that in many shops these kinds of maintenance costs make up 70% of the programming bills.

Rule #1: Get a technically strong and personally forceful person in charge of the Data Dictionary.

The "Data Administrator" is the person who is going to have to tell a lot of people in your company that they can't have what they want, and that their use of a data element conflicts with another use of the element. A person who is too weak to do this can result in your dictionaries and applications becoming complex very quickly. He's also going to have to deal with sneaky programmers who will try to use data elements in strange and undocumented ways; he has to be forceful enough to show them that they can't get away with it.

Rule #2: Use the data Dictionary at the beginning of the development Phase to detect and resolve conflicts in requirements.

If each data element is added to the dictionary as it is defined you can use the dictionary's reports to detect conflicts and in-consistencies. Here's where your strong "data administrator" gets called in to resolve the conflicts, before your programmers go ahead and build the conflicts into the application, or devise some arcane method to get around the conflicts that will work until they leave.

Rule #3: Create and enforce standards and procedures for the Dictionary.

Your staff won't give the dictionary any more respect than you give it; if you make it important and vital, they will follow you. Dictionary updates should be in the hands of the data administrator only; this will give him enough power to deal with the programmers and analysts. Standards for development that ensure that no-one can by-pass the dictionary are also vital to proper maintenance of the developed systems. Ideally, nothing should go into production that isn't fully described in the dictionary.

Rule #4: Retrofit existing systems into the dictionary in order of their estimated lifetime.

"Doomed" systems should go in first, if they are to be replaced. Stable, low maintenance systems have lowest priority, while systems due for extensive maintenance should be dictionarized before the maintenance is started (see rules 2-3 for why this is a good idea.)

APPLICATION PACKAGES:

Rule #5: Let your users select the package they want, from the best ones that you know are available.

With any new application, there will be problems. If your staff is programming the application for the user, the problems will be your problems, affecting your budget and your schedule. If you can avoid being the cause of problems, you'll be able to continue to be of service to your company.

Rule #6: Select a strong member of your group to serve as liason to the user group.

Ideally, this will be someone strong enough to be promoted out of your group and into user management, where he can carry your message into areas that you can't reach, and help build support for you in the various user departments. Remember, your representative speaks with your voice, so don't send a pure technician; send a heavyweight.

Rule #7: Ensure that the users contract with the package vendor for maintenance.

This is important enough to justify rule #6, for users are never happy with the DP's response to maintenance requests. They hate all the paperwork involved. Managing maintenance is a classic "no win" situation — no matter how it's handled, users are never satisfied, so whenever possible, let the vendor handle it. The first task of your "strong" liason person is to convince the user to go with a vendor maintenance contract; let him know that if he fails, he's going to be doing the maintenance for the user.

MICROCOMPUTERS IN END-USER DEPARTMENTS

Many DP shops see Micros as a real menace to the DP department's control of the corporate information resource. They try to make it difficult or impossible for end-users to get and use Micros by blocking any purchases or evaluations of them.

This "dog-in-the-manger" approach to the situtation will be self-defeating in the long run, with serious consequences to the DP department's credibility and stature within the organization. It's far better to encourage end-users to use Micros properly, and to support them in their efforts. You can build your credibility within the organization, while improving the profitablity of the the company by supporting the judicious use of the proper mixture of computer resorces.

Rule #8: Don't fight user-owned Micros — encourage them, instead.

Development of a system is a small part of the total effort involved; maintenance and operations are far bigger pieces of the job. What happens in the long run is that the systems that the users develop on their micros become prototypes for systems that can be developed on the mainframes, if the users have developed a succesful system. If their development effort was a failure, it is buried within the user department; the only systems that the DP department will have to support will be the succesful ones that have outgrown their micro, and need a larger machine to work on.

Rule #9: Encourage your technical people to get involved with micros. This is one rule that it probably will be easy to enforce. Let your programmers help out the users with their technical questions about their micros; encourage your technical people to become familiar with the succesful applications, so that when the time comes to put that application up on the mainframe, they'll know how to do it.

Rule #10: Use micros yourself if you haven't already.

Try out the HP-125; the VISICALC package can make your budgeting easier. You will also get a better understanding of why your users want to by-pass the whole DP department applicatio development process with its paperwork and problems.

APPLICATION GENERATORS:

You already have some manual ones in your shop — the programmers. They can be used to create both simple, repetitive reports or complex applications systems, but they can't do both at the same time. They can also be your liason with the user departments, or write simple, repetitive reports and database update programs. An application generator can free them up to do the things that are more important for the long-range success of the department and the company. It can also shift some of the report-creation into the user departments and out of your budget.

Rule #11: Identify and forecast your programming needs.

If a substantial part of your work is either

A. Ad-hoc report requests or
B. Moderately complicated systems,

then an application generator can ease your burden and reduce your costs. (If your applications needs are simple, you probably should be looking more towards packaged software than to applications generators.)

Rule #12: Select a generator that can be used by both programmers and end-users.

End-users who can produce their own reports using applications generators and who can do their own data entry, are getting what they need from the computer without involving the DP department. Their applications are part of their budget, not yours. Programmers who can use an application generator will typically use it like a ladder, to extend their reach. An application generator that can handle most of the boring detail work reduces the number of programming errors dramatically because boredom causes errors. Application generators can also be very useful with trainee programmers, to take them thru the process of writing a program from specifications by examples.

Rule #13: Select an application generator that is compatible with your Data dictionary.

The application generator should make the maximum practical use of the data dictionary to reduce the drudgery of programming and to control the access to the data in some standardized way. Ad-hoc report generators that interface to the dictionary must provide for separate logical views for each user, as most users have different reporting needs, and do not need extraneous detail.

USING CONTRACT PROGRAMMERS

Many shops are reluctant to use contract programmers but the proper use of contract people can multiply your productivity. One criticism of contract people is that the expertise goes away with them; this criticism conceals an error about what is important. The expertise that is important to the business is the knowledge of the applications, not the knowledge of the computer and

its processes. At any time, you could switch computers, but the business will continue to do what it has been successful at.

This fundamental truth underlays many of the rules that I have suggested so far — that you try to build a staff of people that know the business. This is why contract people can be useful, because you can bring them in to do work that is only technical — language conversions, maintenance, etc, while your stronger staff people work with your users to develop an understanding for and appreciation of the business.

Rule #11: Hire the company, not the programmer.

Find a contracting company that understands your business and your needs and build a long-term relationship with them. You'll soon find that a seasoned professional contractor costs less in the long run than a trainee employee, unless your plan for the trainee is user-support work. Hiring trainees to do maintenance is a sure plan for building plenty of fires to fight, and lots of turnover in the long run.

Rule #12: Use contractors as followers, not leaders.

The best use for contract people is on temporary jobs that have a clearly defined beginning and end; in development projects, they should code programs, set up databases, do documentation, or even be project leaders, but they should not be in charge of defining the user's requirements, as you want to keep the business expertise in-house.

It's also good to put contract people on maintenance work, to reduce your employee turn-over, and the cost should always be billed back to the user department budget. This can be a positive morale-booster for your regular people, who will be getting only the interesting work.If one of your regular employees wants to leave you to become a contractor, with the hope of making big money doing maintenance work, then you're probably better off, because that's not the kind of employee that you should be trying to develop.

PRODUCTIVITY TOOLS

A software tool is a computer program that a programmer uses to do something useful for someone. The first programmer tools were things like COBOL and SORT programs, while current programmer tools include preprocessors that generate programs from pseudocode. The basic thrust is to automate some simple, but repetititous part of the programmer's job, as opposed to the alternative approach of breaking the job down into simple parts and using an assembly-line technique to do the work.

Programmers like software tools because their jobs become less boring; most people are drawn to programming because of the creative aspects of the job, and a properly crafted tool multiplies their power to create. In addition, tools can substitute for other changes in working conditions by providing a work atmosphere that is more pleasant, in the sense of less boring work.

Rule #13: Start with the Data Dictionary.

The role of the Data Dictionary is so central to system development that it is the one productivity tool that you shouldn't try to do without. (see rules #0-#4 for clarification) A properly used dictionary can create savings far in excess of its cost, while reducing development time proportionately. The Dictionary can be so powerful a tool, that the question becomes not "Should I get a Dictionary" but "Which dictionary should I get"?

Rule #14: Get a preprocessor that inerfaces to the Dictionary.

The combination of a dictionary and an preprocessor can reduce your programming effort by 30-70% almost immediately, allowing you to get rid of your backlog of user requests quickly, without hiring additional staff. One very important point is a compiler-data dictionary interface, so that the physical location and characteristics of each data item are taken from the data dictionary at compile-time — which effectively eliminates most maintenance programming.

Rule #15: Enforce structured programming, via a preprocessor or use of a PASCAL-like language.

Structured programming, in the narrowest sense of GOTO-less coding, results in easier-to-maintain code and easier-to-debug applications. Programmers, however, are lazy and will use a GOTO anytime they think themselves into a corner in a program and can't find a simple logical way out. That's one big reason why GOTO-less coding is easier to maintain — programmers don't get to take the easy way out in the first place.

PASCAL and it's relatives are what is being taught in school these days, so using a PASCAL-like language offers the added benefit of not having to train programmers in COBOL, while offering a "state-of-the-art" programming shop as a morale-builder and PASCAL offers the GOTO-less syntax that enforces structured programming painlessly.

CONCLUSION

The set of rules given above is based on the idea that your people, and their knowledge of your company's business, constitute your most important resource. The technical aspects of computing are constantly changing, while the business changes much more slowly. The secret to success is to offer your people an environment where their professional growth is directly related to the growth of the business, and to realize that you can't be promoted until you've found someone who can take your place.