The choice of new software development methodologies for software development projects

by EDWARD YOURDON
YOURDON Incorporated
New York, New York

ABSTRACT

Data processing managers have a number of new "structured" methodologies to assist them in EDP software projects: structured programming, structured design, HIPO, top-down development, structured analysis, structured walkthroughs, and chief programmer teams. Since many of these methodologies are still considered new and "experimental," it is often difficult for the manager to determine which of the methodologies should be used on a software project.

This paper briefly reviews each of the new structured methodologies. It then makes suggestions about the use of the methodologies for new projects, concluding that the use of informal walkthroughs is probably the best way for the manager to introduce the methodologies into an organization that has no previous experience with them.

The point is also made that "research-and-development" projects have different trade-offs than "bread-and-butter" projects. For projects that have hard deadlines and budgets, a number of trade-offs are suggested in order to help the manager decide which of the structured methodologies should be employed.

INTRODUCTION

The data processing project manager of the 1970's has an impressive array of new "structured" methodologies which promise to improve the productivity of his programmers and analysts, as well as improving the reliability, maintainability and overall quality of the finished product.

Unfortunately, there are so many "new" methodologies that the manager may not know which methodology—or combination of methodologies—he should employ on a new project. That choice is made all the more difficult because there is little or no documented evidence to prove the effectiveness of the new methodologies. Indeed, the problem is even worse: a variety of exaggerated claims in the popular EDP trade journals has made many a manager so skeptical that he may be unwilling to experiment with any of the new methodologies.

The purpose of this paper is to provide some useful advice to the project manager who finds himself in this position. A brief description of the more widely-known "structured" methodologies is given; following that, some suggestions are given as to the sequence and manner in which the new techniques can best be employed.

AN OVERVIEW OF THE STRUCTURED SOFTWARE DEVELOPMENT METHODOLOGIES

The collection of new software development methodologies is sometimes referred to as PPT (Programmer Productivity Techniques), or IPT (Improved Productivity Techniques), or SPT (Structured Programming Techniques). Almost every data processing organization has a slightly different understanding of the specific techniques which comprise the overall collection; however, the most common ones appear to be the seven described below.

Structured programming

Structured programming is widely regarded as the "first" new development methodology. Based on some theoretical work by Dijkstra, Böhm and Jacopini in the mid 1960's, it has recently been discussed in a number of textbooks and literally hundreds of papers in the computer literature.

A number of people have begun using the phrase "structured coding" to emphasize this methodology's most important features: procedural logic based on combinations of IF-THEN-ELSE, DO-WHILE and "sequence" structures. Such logic usually eliminates the need for GOTO statements, or unconditional branching instructions—a fact which has resulted in structured programming being given the nickname "GOTO-less programming."

Proponents of structured programming claim that structured code is easier to comprehend; it therefore tends to be more maintainable, and is more likely to be correct code. It is generally agreed that structured coding adds an overhead of 5-10 percent to the memory requirements and execution time of the program; however, there have been several cases where structured code has been more efficient than unstructured code, simply because it is better organized.
Structured design

Structured design is usually considered a "newer" discipline than structured programming, even though many of its concepts have been discussed by EDP professionals for several years. A 1974 paper in the IBM Systems Journal\textsuperscript{19} marks the beginning of "real" interest in the subject; that paper was quickly followed by a number of books\textsuperscript{11-14} and the usual plethora of papers in the popular journals and conference proceedings.

Structured design is usually described as the process of deciding which modules, interconnected in which way, will best solve some well-stated problem. Its emphasis is on techniques for identifying "good" modules (good, that is, from the viewpoint of maintenance and modification, rather than execution speed or memory requirements), and on systematic "cookbook" methods for deriving "good" designs for common types of EDP problems.

Proponents of structured design claim that it has the same virtues as structured programming: greater reliability, improved maintainability, and greater comprehension of how the system works. Some limited experiments suggest that structured design has a far greater impact on maintainability than structured programming, since it concentrates on building systems from small, highly-independent, single-purpose modules.

HIPO and other documentation techniques

Along with the recent interest in structured programming and structured design, there has been a great deal of interest in some new documentation techniques which can help describe the procedural logic represented by structured programming, and the architectural design represented by structured design.

The most widely known documentation technique is known as HIPO—an abbreviation for "Hierarchy, plus Input, Process and Output." Originally developed by IBM, it has recently been described by Katzan\textsuperscript{15} in a sufficiently thorough fashion for EDP managers to consider using it as the documentation standard on new projects. An alternative diagramming technique, known as "structure charts," is described by Yourdon and Constantine.\textsuperscript{12}

To document procedural design—that which has classically been documented with flowcharts, decision tables and narrative English text—such techniques as pseudocode (also known as "program design language")\textsuperscript{19} and Nassi-Schneiderman diagrams\textsuperscript{17} have gained popularity in some organizations. Other organizations have abandoned detailed documentation altogether, feeling that any method of detailed documentation will suffer from the problem of obsolescence.

Top-down development

Many data processing organizations were introduced to the concept of "top-down design" at the same time they were introduced to structured programming. This kind of design approach has also been referred to as "stepwise refinement," or "levels of abstraction," or "divide and conquer." Only recently has it become evident that one can easily design a bad system in a top-down fashion.

Meanwhile, there has been a great deal of discussion about the manner in which one should implement a well-designed system. In contrast to the classical approach (now referred to as "bottom-up") of unit testing low-level modules, and then integrating them into larger entities, it is now becoming popular to work in the other direction. That is, the "top-down" approach to implementing systems requires the coding and testing of the top-level (or "executive") module first, with the lower-level modules taking the form of "stubs" (a typical example of a stub is a module which exists immediately without doing any real processing.) Subsequent development of the system involves the substitution of real modules for the stubs.

Proponents of top-down implementation claim that it has a number of benefits, many of which are "political" in nature. The top-down approach tends to distribute system testing and integration throughout the entire project, rather than saving it for the end of the project. It also tends to expose major interface problems early in the project, rather than leaving them until the end of the project. Equally important, the top-down approach usually allows the project manager to demonstrate a working subset of the system to the customer at an early date, and the existence of a working subset also allows him to survive deadline crises more gracefully. It has also been observed that the top-down approach tends to distribute the requirements for testing resources more evenly throughout the project; by contrast, the "bottom-up" approach to testing usually requires large amounts of testing resources (e.g., computer test time) toward the end of the project—and it may be physically impossible to schedule, say, 25 hours per day of computer test time.

Structured analysis

With the advent of structured programming and structured design, it became clear that the major unsolved problem was that of the user: specifically, the problem has been that of figuring out what the user wants, so that a good system can be designed (using structured design), coded (using structured programming), implemented (using top-down development), and documented (with HIPO or structure charts).

Structured analysis addresses this problem. Its basic objective is to provide a formal description of the user's requirements, expressed in logical terms (i.e., with as little reference as possible to the peculiarities of a specific machine, a specific data base management system, etc.), using standard tools and building blocks. Its key ingredients are communication tools to improve the communication between analyst and user, and a new approach to the "system's development life cycle" that encourages both user and
structured walkthroughs

The concept of walkthroughs, or "code reviews," seems to have its historical origin in Gerald Weinberg's classic book, The Psychology of Computer Programming. Since then, it has been discussed in a number of places and is regarded one of the more important of the new structured methodologies. In their simplest form walkthroughs are a somewhat informal procedure for reviewing the correctness and quality of the analysis, design, code, test data and documentation associated with a software project. The review is normally carried out by the programmer's peers, rather than his supervisors; indeed, the review is normally done by all of the members of the project team.

Proponents of walkthroughs claim a number of benefits; increased reliability of the delivered product; more comprehensible and maintainable code; greater learning and sharing of information among team members; and a greater chance that a partially completed program can be salvaged if a programmer leaves in the middle of the project.

Chief programmer teams

Originally referred to as the "superprogrammer team" in the mid 1960's, the chief programmer team concept first attracted wide-spread attention on the New York Times project, where it was used by IBM in conjunction with structured programming, top-down implementation and a variety of other techniques. Since then, it has been discussed in a variety of publications.

The basic concept of a chief programmer team is to organize a software development project around a person who has (a) programming abilities substantially greater than—e.g., an order of magnitude greater than—other programmers in the organization, (b) ability to provide the documentation for the code, the operational procedures and the user manuals for the system, and (c) the ability to supervise a team of specialists include a "copilot" (an apprentice chief programmer), a "language lawyer" (an expert in the programming language or operating system or data base management system being used), a "toolsmith" (a person who develops useful debugging packages or other software development tools for the specific use of the project), a "librarian" (a person who organizes and controls the source programs, object programs, listings, and other documents associated with the project).

Proponents of the chief programmer team approach point out that it is merely taking advantage of some well-known facts about differences in programmer abilities. In addition, they point out that the concentrated talents of one superprogrammer makes it possible for a medium-sized software development project to be accomplished with a much smaller group than would otherwise be necessary; consequently, the project manager can expect far fewer communication problems than he might otherwise expect.

SUGGESTIONS FOR INTRODUCING THE NEW STRUCTURED METHODOLOGIES

Unfortunately, it is not possible to give a simple algorithm in this area. We cannot easily say, "First you should introduce structured programming, then you should use structured design," nor can we say, "If you are working on a payroll system, then you should definitely use chief programmer teams; on the other hand if you are developing a real-time telecommunications system, you should use only structured walkthroughs."

On the other hand, the structured methodologies have been introduced into enough organizations that we can draw some general conclusions from their experiences. These are given below.

Trying to implement all of the new structured methodologies at once will generally be a disaster

Some organizations can actually pull off such a feat. After reading about the new methodologies, or getting a presentation from their friendly hardware vendor, they decide to use all of the new methodologies at once. As one might expect, this is more likely to happen in the smaller EDP organizations—those with only half a dozen programmer/analysts—and is not very likely to occur in the larger organizations.

Sometimes, though, an organization will decide to try all of the new structured methodologies on a single project; this is quite common when the organization decides to use the new methodologies as an experiment in a so-called pilot project. Even in a limited situation like this, it usually turns out that an attempt to experiment with half a dozen new methodologies at once leads to chaos and confusion.

The reasons are obvious enough. Structured programming and structured design are not simple concepts, and a lot of concentration is needed to make them work right. If the programmers are also trying to implement walkthroughs—which require a great deal of psychological energy, too—and chief programmer teams, as well as adjusting to the concept of a librarian relieving them of their clerical work . . . well, it will be a wonder if they get any of it right.

Techniques which involve organizational change are often the most difficult to implement

Some organizations will find it difficult to ever implement chief programmer teams, librarians and walkthroughs. The
point here is that even if the project manager can convince his organization to try the chief programmer team concept, or librarians, or walkthroughs, he will probably find that difficult as his first new methodology. The author's experience has been that it is somewhat easier to introduce a relatively innocuous technical concept like structured programming first—that doesn't threaten anyone's empire, and is not likely to be at odds with current organizational philosophies.

Once the project manager has demonstrated that structured programming, top-down implementations and structured design are good ideas, then he'll probably be in a strong enough political position to say to the big boss, "Listen, the last three structured methodologies that I introduced to the company turned out to be winners. Why not gamble a little now, and let me try something like the chief programmer team concept?"

Structured code without structured design is often worthless

A number of organizations have found recently that structured programming (or, more specifically, structured coding) is a great idea but that it is not enough. If the modules in an EDP system are too large, too complex, and too interconnected with one another, then maintenance problems will persist regardless of the presence or absence of GOTO statements.

This raises some interesting political consequences. If the EDP organization has been doing things in a backwards fashion for years, and if the project manager introduces the new structured methodologies with great fanfare and promises of spectacular improvements, then the first new methodology should indeed demonstrate spectacular improvements.

And if the project manager tries structured programming alone, he might not achieve such spectacular improvements. The author's experience on a few EDP projects lately has been that the initial productivity and reliability will seem quite impressive, but the long-term maintainability of a system produced with nothing more than structured coding may not be very impressive at all.

The moral: It may make good sense to begin with structured design first—and when that is working properly, then introduce structured coding. Once the project manager has overcome all of the objections and battles and problems associated with structured design, it will be almost trivial to introduce structured programming.

There is a more important reason for this suggestion: good design and mediocre coding is a tolerable state of affairs; mediocre design and good coding, on the other hand, is not a good formula for success. And if the project manager thinks that his project team has energy, intelligence and enthusiasm to tackle only one new methodology, then structured design should get preference over structured coding.

Top-down design and implementation are often a good way of introducing the new structured methodologies

It is frequently observed that many of the benefits of top-down implementation are "political" in nature. It allows the project manager to demonstrate a working subset of his system to the user at an earlier point in time; it allows him to survive deadline crises more gracefully; and it allows him to schedule testing resources (e.g., computer test time) in a more manageable fashion.

These benefits are very noticeable to the user community, to higher levels of management, to the computer operations manager, and to various other people in the organization. For that reason alone, many EDP managers have decided that the top-down approach is a good way to introduce the new structured methodologies in their organizations.

Keep in mind that this approach can backfire. Unfortunately, many programmers view top-down implementation as an invitation to begin coding before they have done any real design. Especially on the first few projects, the manager should beware of this danger.

The most successful approach has often been informal walkthroughs

There is a strong argument for informal walkthroughs as the project manager's first venture into the new structured methodologies. Note the emphasis on "informal" walkthroughs—not necessarily with all the "bells and whistles" that are normally suggested (see, for example, the detailed procedures suggested in Yourdon's "Standards for Structured Walkthroughs" [23]).

Why would informal walkthroughs be a good way to get started with the new structured methodologies? For the simple reason that the project manager can't trust any individual programmer to understand and implement any of the other methodologies by himself. By forcing everyone to talk about their designs and their code—in an informal, low-key, non-threatening fashion—the manager can maintain some kind of quality control when he most needs it.

This is a point that needs emphasizing. If the manager has 30 programmers, and if he gives them all the standard textbooks on structured programming, they are almost guaranteed to read 30 different (and almost mutually exclusive) things. They will write 30 different kinds of structured programming—some good, some mediocre, and some downright bad (indeed, probably even worse than the kind of code that was written before structured programming came along). And if nobody looks at their code (which is the current state of affairs), the manager will never know who really understands structured programming, and who doesn't.

If the project manager begins by establishing an environment of exposing everyone's code to public discussion, then he will ensure that a relatively uniform version of top-down implementation, structured design, and structured programming can be implemented later on.
CONCLUSION

In the final analysis, only the project manager can decide which of the new structured methodologies he wants to introduce on a project. The suggestions in this paper can do nothing more than make the manager think about trade-offs that have been observed in other EDP projects; it is up to the manager to apply those trade-offs to his own project.

One of the most important questions the manager must ask himself is whether the new structured methodologies should be considered as a set of experimental "R&D" concepts, or whether they are to be considered down-to-earth practical concepts, with an immediate payoff.

Indeed, some organizations deliberately use the new structured methodologies on experimental "pilot" projects, with no preconceived ideas about which ones will work and which ones won't. In such an environment, the manager should use any and all of the methodologies that are of interest to him; our only caution is to arrange the pilot project in such a way that the impact of each new methodology can be measured in some crude fashion.

If the manager is involved in a "real" project—with real deadlines, real budgets, real users with real needs, and real penalties if the project fails—then he should be considerably more cautious about the new methodologies he employs. In this case, he will have to take into account his own perceptions about such things as:

a. The political climate within his organization—will the manager be given any encouragement if it is seen that he is "experimenting" with new technologies?
b. The nature of his project team—are they enthusiastic enough, experienced enough, and bright enough to try new methodologies while simultaneously working against a real deadline and budget?
c. The "learning curve" of the new methodologies—even with the best group of programmer/analysts, some time will be required to begin using the new methodologies properly. Is the project large enough and long-living enough to accommodate an initial investment in "learning" in return for a long-term payoff in productivity, reliability and maintainability?
d. The perceived "payoff" of the new methodologies—are they really as good as the popular EDP journals say they are? The manager has to make his own judgment of the impact on structured design, structured programming and walkthroughs on his project; this may be influenced by the nature of the applications, and various other factors.
e. The alternatives—if the manager elects not to use the new methodologies, what else can he use? On a simple EDP project, the manager may decide that the project is guaranteed to fail with the conventional methodologies; in such a situation, the manager may decide to "go for broke," and try all of the new methodologies.

REFERENCES
