Understanding the developmental life cycle

by RAY CAUDILL
Air Force Data Automation Agency
Gunter AFB, Alabama

ABSTRACT

The paper supporting the topic above asserts that the software development manager is faced with a myriad of technical questions to which he must seek technical solutions. The paper makes a case for applying an adaptation of configuration management to the software developmental life cycle. This is done by first stating that the life cycle can, in fact, be divided into reasonably discrete parts even though the developmental life cycle is a continuum. The paper then gives an overview of Automated Data System Project Management, highlighting phases, reviews, deliverables, baselines and scaling. An alternative to this version of configuration management is also offered for the purpose of showing the flexibility of configuration management. The paper concludes by asserting that the elusive term "management visibility and control" becomes realistic and achievable through application of configuration management. The paper concludes by asserting that the elusive term "management visibility and control" becomes realistic and achievable through application of configuration management. Also, a better choice of tools and techniques can be made when gauging technical need against phases rather than the entire developmental life cycle as a whole. Lastly, the paper concludes that progress measurement and management reporting actually can be achieved through documentation of the results of reviews of deliverables expected from each phase.

INTRODUCTION

The title of this session is "Software Management: How to Start a Software Development Project." This is one of the less technical sessions and is aimed at the manager, new or otherwise, who is about to begin a development effort. A senior analyst/programmer, given project leadership for the first time, is confident of success. He's been there. He's done the "grunt" work. He knows how to "build" software. However, he has never "managed" a software development effort. There is a difference. His confidence in his eventual success transcends his superiors and staff alike. Because of all this show of confidence, everyone predicts high marks for this project to be finished on time, within budget estimates, and that it will totally satisfy the customer. Amid all this optimism, the manager is asking questions like: What skills do I need?; Which of the many tools and techniques on the market are right for me?; and, How do I know my plan is holding? In short, he's doing what most people think he should be doing. That is, he is seeking technical solutions to technical problems. He just may be making his first serious mistake. He may be beginning on an elevated technical plane much too quickly.

My suggestion is that he hold the technical questions a little longer and ask and seek a solution to a less technical one first. He should ask himself this all-too-rare question first: "What are the Component Parts of this thing called the Software Developmental Life Cycle?" If he can see through or past the "whole" and actually discern the smaller, somewhat discrete, parts of the developmental life cycle, and plan and manage accordingly, he will possess and be able to articulate a much greater understanding of the whole. I firmly believe, then, that he has a much greater chance of a level of success that just might match his initial optimism.

Before proceeding, let's briefly explore some of the reasons why software development efforts fail anyway. You each have your own opinions, some of which relate to technical complications and perhaps to the employment of specific tools and techniques. However, I'd like to make some of my own observations as to why failures occur.

Lack of management visibility and control

Management at all levels "assumes" that the manager closest to the development effort does have visibility and control of the total effort and that a simple query will bring a clear and concise response as to project planning and progress. Some project managers do have such visibility and control, but there is absolutely no reason for you or anyone else to assume they do. If your thought and planning processes don't or can't transcend the "whole" and see the parts of the software development life cycle (which is the only place where visibility and control exist) then why should you expect another manager, technical or otherwise, to do so?
Pressures from higher authority

Here's a serious problem for most of us. Let's say you've laid out a positively foolproof developmental plan. Higher management then decides to step up your schedule by six months, or to reduce your budget by some large amount. Well, you have no choice now but to modify that great plan you had. You guessed it, it's that alternative plan that is doomed to failure from the start. But why should that be so? If your plan is by phases, or parts of the whole, you can modify each accordingly and the basic plan still holds. You can't just cut out and throw away, or ignore a part of the developmental cycle just because of a reduction in time, people, or money. To say it another way, there are certain actions or activities which must occur in order to develop a complete computer software system. You cannot cut out any one of those activities; you can only develop less system. The alternative (the one doomed to failure) is to develop the same system but with less quality.

Now for a couple of "truisms" or "propositions" to cement my position for you and to give you a baseline, or point of departure for the remainder of this paper.

Proposition 1: The developmental life cycle is a continuum. Dictionary Definition: A continuum is a "succession, or whole, no part of which can be distinguished from neighboring parts. . . ."

Proposition 2: The developmental life cycle is a succession of discernible, manageable, and measurable parts.

These propositions appear contradictory. I happen to believe both and hope to convince you of the same. More of us than care to admit have been involved in failures because we saw only the "whole" and managed accordingly rather than seeing through the "whole" to its "parts" and dealing with them on an individual basis.

PHASE RECOGNITION

There are as many phases in the developmental life cycle as you'd like there to be. I'll show you two examples of phase use later, but for now just visualize your entire project as having the following six phases:

1. The idea and its approval.
2. Refine the idea into a user requirement.
3. Design the application system.
4. Develop, code, and test programs.
5. Total system testing.
6. Operate the system.

It is only after one recognizes these "parts," now referred to as "phases," that the technical questions referenced earlier take on some perspective. You're able now to apply questions to specific phases of the software development life cycle and deal with them accordingly. Also, those rather elusive terms such as "management visibility and control" begin to take on true meaning. Planning each phase singly, then putting them together can make for a more detailed and a more accurate plan. You've all witnessed this negative example of project visibility and control. A project is assigned to a group of people with the singular guidance: "Have it on the air in 18 months." You, and they, are quite naturally 1/18 more nearly complete at the end of every month. Suddenly, in the 16th or 17th month, you find that you/they are only 50 percent finished. You're forced to accept that assessment even though you're too embarrassed to ask the question: "50 percent of what?"

THE ADVANTAGES OF PHASES

There are many advantages to development by phases:

Here are a few:

- Each phase has a relationship to all others and to the entire project (the "one small step at a time" concept).
- Comprehension of each makes the "whole" comprehensible.
- Each phase must yield some deliverable product in order to terminate.
- User/developer reviews of deliverables for technical acceptability become the measure of progress and the method of reporting that progress to management.
- Responsibilities/accountability for specific phases, and deliverables can be identified.
- User or design changes can be controlled and accounted for.
- Finally, dissecting the developmental life cycle into deliverable-oriented phases with appropriate reviews can:
  - Provide early warning of technical problems/slippery pages.
  - Provide setting for controlled user/developer relations.
  - Provide for some realistic assessment of where the effort stands relative to plans.
  - Make visibility and control realistic.
  - Enable the development manager to respond to serious changes directed by higher management echelons.
  - Lessen the trauma of implementation.

CONFIGURATION MANAGEMENT

Of course, what I'm leading up to is a set of concepts and principles commonly referred to as configuration management. For years, configuration management has been the tool of managers of ultra-large efforts such as shipbuilding and aircraft manufacturing. In the last few years, however, more and more people are applying the same concepts and principles, with modifications, to software development. My first thought, upon reading about configuration manage-
ment for the first time, was that it is overkill to the nth degree. It is only if you allow it to be. The concepts and principles have to be tailored and tempered to meet the requirements and scope of each project. Figure 1 is a simplistic definition of software configuration management and shows the aspects of it this paper will develop. Those who have dealt with configuration management in the development of weapons systems may believe that we have strayed rather far from what you know as configuration management. I do not believe we have. Those of us who developed the version I'll touch on were always mindful of phasing, deliverables, reviews, etc.

AUTOMATED DATA SYSTEMS PROJECT MANAGEMENT

We are using configuration management now in software development. I believe it is safe to say we have had developmental successes and a failure or two. With no particular case in mind, perhaps we would have had fewer failures plus more efficient successes had we become acutely aware of the potentialities of configuration management many years ago. There are various elements within the software development field which have been exploring and in fact implementing configuration management for the last couple of years. Unfortunately, it takes considerable time to get a process of this magnitude legislated and in use across the entire organization. We are now on the threshold of the broadest possible usage of the concepts and principles of configuration management. I'll spend the next few moments giving you an overview of our approach to it. In the interest of brevity, it will be precisely that, an overview. Figure 2 shows the portions of the document on this subject which will be discussed in this paper.

THIS DOCUMENT CONTAINS OUR APPLICATION OF CONFIGURATION MANAGEMENT TO DEVELOPMENT OF COMPUTER SOFTWARE.

HIGHLIGHTS TO BE DISCUSSED

PHASES
DELIVERABLES
REVIEWS
BASELINES
SCALING

Figure 1—Configuration management

Figure 2—Automated data systems project management

We recognize these phases: Conceptual, definition, development, test and operation. Figure 3 shows graphically the phases and what occurs during each. Here follows a brief narrative description of each:

Conceptual phase

The purpose here is to identify and fully analyze in detail the user requirement. This also involves project planning and gaining approval for project development.

Definition phase

During this phase, the total system is designed and cross-checked to assure that each system function satisfies a user requirement stated in the previous phase.

Development phase

All programs are designed, coded and tested during this phase. Various reviews are also accomplished to assure technical acceptability of the programs and to assure that products produced are as intended.

Test phase

The total system is tested in an independent environment as well as in the user environment if necessary during this phase. Audits are performed here to ascertain user acceptability of each deliverable product.

Operation phase

The system is put into full operation. Feedback from the user is as required to gain from his experience.

PHASE DELIVERABLES

No attempt will be made here to list or discuss all deliverable products for each phase. The purpose here is to give you some insight into the type of deliverable product that is prepared and reviewed in each phase.

Conceptual phase

Data Automation Requirement—This is a statement of a problem needing attention and will include as much relevant detail about the problem as is available at the time.

Functional Description—This is the most detailed statement of the user/customer requirements. This is the docu-
<table>
<thead>
<tr>
<th>CONCEPTUAL</th>
<th>DEFINITION</th>
<th>DEVELOPMENT</th>
<th>TEST</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENTIFY AND ANALYZE THE USER REQUIREMENT</td>
<td>DESIGN THE TOTAL SYSTEM</td>
<td>DESIGN PROGRAMS, CODE AND TEST</td>
<td>TEST TOTAL SYSTEM IN INDEPENDENT AND USER ENVIRONMENT</td>
<td>OPERATE</td>
</tr>
</tbody>
</table>

Figure 3—Phases

**Definition phase**

System/Subsystem Specification—This “document” is the system design and includes such items as the total system design, interfaces with other systems, subsystem specifications, etc.

**Development phase**

Program Specifications—This document includes the logic of each computer program and is the basis for coding. It specifies the functions to be performed by the program, its inputs/outputs, and interfaces.

Coded Programs—Self explanatory.

Test Plans—As the name implies, this document includes plans for testing the total system independently and in the user environment.

Operator and User Manuals—These documents include everything the operator needs to know to operate the systems, and everything the user needs to know to “use” the software in his environment.

**Test phase**

Test Reports—The results of all test activities are recorded and fed back to the developers and the users.

**Operation phase**

User Feedback—There is an administrative procedure established to cover user feedback. However, letters, phone calls, and visits are all used for this purpose.

**REVIEWS**

There are reviews that must be accomplished throughout the software developmental life cycle. Figure 4 shows some of the reviews we have elected to use. Reviews are iterative. That is, if the product reviewed does not fulfill all requirements, it’s back to the drawing board with suggested or directed changes. Also, it is important to point out that the user/customer is involved to some degree, as a direct participant, or as an observer, in virtually every review. Hence, the user is always apprised of progress and he always knows exactly what he’s going to get. Secondly, if he changes his mind about his requirements late in the cycle, he has full understanding and appreciation that it
may cost him in terms of his desired on-line target date. However, by keeping him involved, he’s more likely to give you well thought-out requirements to begin with. He’s less likely to say what so many have said in the past—“just give me something and I’ll tell you whether I like it or not.”

Conceptual phase

*Systems Requirements Review*—Through this review, the user and the developer reach a common understanding of the user requirements, of conceptually what is to be developed, and of the plan for doing so.

Definition phase

*System Design Review*—The purpose of this review is to gain both user and developer approval of the total system design and to assure that the design satisfies the requirements approved in the previous phase.

Test phase

*System Validation Review*—The purpose of this review is to ascertain that the system performs the function for which it was designed.

There are various audits which occur during the Develop-

---

<table>
<thead>
<tr>
<th>CONCEPTUAL</th>
<th>DEFINITION</th>
<th>DEVELOPMENT</th>
<th>TEST</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENTIFY AND ANALYZE THE USER REQUIREMENT</td>
<td>DESIGN THE TOTAL SYSTEM</td>
<td>DESIGN PROGRAMS CODE AND TEST</td>
<td>TEST TOTAL SYSTEM IN INDEPENDENT AND USER ENVIRONMENT</td>
<td>OPERATE</td>
</tr>
<tr>
<td>SYSTEM REQUIREMENTS REVIEW</td>
<td>SYSTEM DESIGN REVIEW</td>
<td>PRELIMINARY DESIGN REVIEW</td>
<td>SYSTEM VALIDATION REVIEW</td>
<td></td>
</tr>
<tr>
<td>CRITICAL DESIGN REVIEW</td>
<td>PURPOSE: REVIEW/APPROVE PROGRAM SPECIFICATIONS PRIOR TO CODING</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4—Reviews**

*NOTE:* THIS CHART REPRESENTS ONLY A SELECTED SAMPLE OF REVIEWS WHICH CAN BE USED.
ment and Test phases. These are generally conducted with the user as a final physical check on all deliverable products. Audits usually serve as a portion of the "grunt" work that goes into the System Validation and Product Verification Reviews. Secondly, all reviews are accomplished with the rationale that each product reviewed must be traceable back to some previously reviewed and approved document.

BASELINES

A baseline serves as a point of departure for future development or action. Hence, once the system design (system/subsystem specification for example), has been reviewed and approved, it is baselined. All future program development must satisfy that baselined document. This isn't to say that changes are forbidden. They certainly are not, but they are controlled and are subject to the same reviews and approvals as were the original documents. Baselines are generally established at the conclusion of a phase, but there is no reason a specific document cannot be baselined anytime that it suits your plans. The point I wish to make is that baselines are flexible, just as any other principle of software configuration management is. You must determine for yourself what your baselines are to be. That means you must determine what documents are to be baselined, when they are to be baselined, and by which review. This aspect alone forces you into some pretty finite project planning.

PROJECT SCALING

I never intended to make you think that every project undergoes all these reviews, and must produce all the documents. Remember the statement about "overkill" that I made earlier in this paper. For smaller projects, reviews can be combined or eliminated and so can documents. However, this is done only after adequate planning. I'll provide only one example which gives you an idea of how we would scale the procedures of configuration management to a specific project. This example shows what reviews will be accomplished for a project based upon project manyears. The following Decision Logic Table is self-explanatory and happens to be for projects which would be developed for multiple site implementation.

<table>
<thead>
<tr>
<th>Project Scaling (Example)</th>
</tr>
</thead>
<tbody>
<tr>
<td>If Project Manyears are: 1.&lt;10 10.&lt;25 25.&lt;50 50-&gt;</td>
</tr>
<tr>
<td>Then Perform:</td>
</tr>
<tr>
<td>System Requirement Review</td>
</tr>
<tr>
<td>System Design Review</td>
</tr>
<tr>
<td>System Validation Review</td>
</tr>
<tr>
<td>Product Verification Review</td>
</tr>
<tr>
<td>Preliminary Design Review</td>
</tr>
<tr>
<td>Critical Design Review</td>
</tr>
<tr>
<td>Audits</td>
</tr>
</tbody>
</table>

ANOTHER EXAMPLE OF APPLIED SOFTWARE CONFIGURATION MANAGEMENT

My purpose is to convince you that your chances of success are probably greater if you accept the concepts and principles of configuration management and tailor them to suit your situation. With that in mind, I'd like to show you another variation of what I've just shown you in the last several charts. This variation uses the same phase names but with different accomplishments in each. It does not use the same reviews either, but a study of the document from which it came revealed that the same activities generally occurred. The following comparative chart will show you the differences between these two variations.

<table>
<thead>
<tr>
<th>APPROACH #1</th>
<th>COMPARATIVE PROCEDURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCEPTUAL</td>
<td>CONCEPTUAL</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>DEFINITION</td>
</tr>
<tr>
<td>DEVELOPMENT</td>
<td>DEVELOPMENT</td>
</tr>
<tr>
<td>TEST</td>
<td>TEST</td>
</tr>
<tr>
<td>OPERATION</td>
<td>OPERATION</td>
</tr>
</tbody>
</table>

Figure 5 shows a comparative example of phasing along with a brief statement as to what occurs in each phase. This comparative procedure was included to show an example of how software configuration management can be varied. The same phase names were used in both the versions above, but the activities which occur in the first three are quite different. For example, in our version, the Conceptual Phase covers everything from the initial idea, project approval through the detailed user requirements definition. The comparative version uses the Conceptual Phase to deal only with the basic idea and the approval for its development. The Definition Phases in the two versions also vary considerably. In our version, the system is designed, whereas in the other, definition means detailed definition of
Figure 5—Another example of phasing

the user requirements. There are other differences as well. The point I wish to reiterate is that Software Configuration Management is flexible and should be tailored to suit your needs.

SUMMARY

The introduction to this paper asserts that many technical problems in need of technical solutions beset the manager as he contemplates a developmental effort. However, this paper further asserts that the obvious technical questions can best be addressed if put into proper perspective. A way of doing that is to deal with the whole (developmental cycle) by separating it into its component parts or phases. We have adopted the concepts and principles of configuration management toward that end. The example of another variation of configuration management was briefly discussed to assure you that you have much latitude in applying the concepts and principles of configuration management.

CONCLUSION

The developmental process is a continuum. However, I have concluded that the application of the concepts and principles of Software Configuration Management should be given priority consideration when beginning a software development effort. Planning, controlling and developing then become phase/review/deliverable oriented and progress measurement and reporting become realistic and meaningful. Specific advantages of applying configuration management to the software development life cycle are:

- Planning, controlling and visibility become more realistic because reviews of specific deliverables must be accomplished throughout the cycle.
- Plans which have been prepared around the phased concept are easier to change.
- Technically oriented tools and techniques can be selected and applied to phases or groups of phases with more confidence.
- A workable user/developer relationship can be identified and controlled.
- The biggest pay-off of all: — We data automation people can now "articulate" to others precisely what the developmental life cycle actually is.
- Greater confidence from management and users.
- System developers' credibility can be enhanced.
- Can stop or at least impede the "rush to the code sheet."
- Can articulate the need for a somewhat greater investment up front which will increase both speed and accuracy of the systems work and program development.