Systems Development Life Cycle:

Objectives and Requirements

Systems Development Lifecycle:
Objectives and Requirements is

Bender RBT Inc.
17 Cardinale Lane
Queensbury, NY 12804
518-743-8755
rbender@BenderRBT.com

Chapter 1 Introduction 4
Chapter 2 Executive Summary 5
Chapter 3 Why An SDLC? 6
Chapter 4 SDLC Objectives 8
4.1 Ensure High Quality 8
4.2 Provide Strong Management Control 9
4.3 Maximize Productivity 9
Chapter 5 SDLC Requirements 11
5.1 Scope Requirements 11
5.2 Technical Activities 14
5.3 Management Activities 30
5.4 SDLC Usability Requirements 42
5.5 Installation Support Requirements 43
Chapter 6 SDLC Design 45
6.1 An Example Of Itself 45
6.2 A Layered Approach 46
6.3 Delineate What From How 52
6.4 Hierarchical Structure 53
6.5 SDLC Automated Support 53
Chapter 7 SDLC Project 53
COMMON SDLC QUESTIONS 55

Chapter 1 Introduction

This document has two objectives. The first is to explain what a systems development lifecycle (SDLC) is and why one is needed. The second is to provide a yardstick for evaluating any SDLC, whether developed internally or supplied by a vendor. This latter objective will be met by clearly defining the SDLC's objectives and requirements. Key design approaches for meeting the objectives and requirements will also be discussed.

Chapter 2 Executive Summary

A systems development lifecycle (SDLC) has three primary objectives: ensure that high quality systems are delivered, provide strong management controls over the projects, and maximize the productivity of the systems staff.

In order to meet these objectives the SDLC has many specific requirements it must meet, including: being able to support projects and systems of various scopes and types, supporting all of the technical activities, supporting all of the management activities, being highly usable, and providing guidance on how to install it.

The technical activities include: system definition (analysis, design, coding), testing, system installation (e.g., training, data conversion), production support (e.g., problem management), defining releases, evaluating alternatives, reconciling information across phases and to a global view, and defining the project's technical strategy.

The management activities include: setting priorities, defining objectives, project tracking and status reporting, change control, risk assessment, step wise commitment, cost/benefit analysis, user interaction, managing vendors, post implementation reviews, and quality assurance reviews.

In order to meet all of the SDLC's objectives and requirements there are certain design approaches that are required: the SDLC must be an example of a system created using the techniques it espouses; it must use a layered approach to analysis, design, installation support and production support; it must keep distinct the "what" from the "how" in regards to doing the tasks and creating the outputs; and it must organize its information in a hierarchical manner so that users with varying degrees of familiarity can find what they want easily and quickly.

Defining or selecting an SDLC should be undertaken as a project with full time resources who have the appropriate level of expertise. It is an extremely high leverage effort. It also represents a major cultural change for the staff. It must be planned and executed in as professional a manner as possible.

Chapter 3_Why An SDLC?

When you pay your telephone bill your payment is processed by a system. That system has evolved over many years and continues to evolve in order to meet the changing needs of the business. When the phone company cashes your check that check is also processed by a system which itself is evolving. These two systems are composed of manual activities and automated components. They also exist in the context of many other systems with which they must interface.

Each system works so well individually because it is composed of a rigorous set of tasks which result in well-defined outputs. Regardless of who is doing the task, the result is essentially the same. As staff turnover occurs the system provides continuity to the way of doing business.

Each system can interface with the other because the division of activities between the bank and the phone company are well defined, as are the interfaces. Thus, no matter which bank the check is drawn on, the process is the same; no matter which phone company sends in the check, the process is the same.

The precision and completeness of the task lists, the data, the division of responsibilities and the interface definitions are required because of the complexity of these systems. But what about the process that creates and maintains these systems?

Software systems development is, from a historical perspective, a very young profession. The first official programmer is probably Grace Hopper, working for the Navy in the mid-1940s. More realistically, commercial applications development did not really take off until the early 1960s. These initial efforts are marked by a craftsman-like approach based on what intuitively felt right. Unfortunately, too many programmers had poor intuition.

By the late 1960s it had become apparent that a more disciplined approach was required. The software engineering techniques started coming into being. This finally brings us to the SDLC.

What evolved from these early activities in improving rigor is an understanding of the scope and complexity of the total development process. It became clear that the process of creating systems required a system to do systems. This is the SDLC. It is the system used to build and maintain software systems.

As with the phone billing system, an SDLC system is needed because the development process is composed of many complex tasks which must be done in the right order to produce a successful result. If there is no SDLC each team must reinvent it based on their own experiences and judgments.

The complexity of the tasks has led to increased specialization. These specialists (e.g., data base analysts, network designers, testers) must have well-bounded tasks with well-

defined outputs and well-defined interfaces to the rest of the development team. This is provided by the SDLC.

Systems also have a long life. The billing system and check processing system will probably never be de-automated. These systems will outlive their development teams. As staff turnover occurs, continuity is required in how the systems are supported. This is provided by the SDLC.

Another issue is systems integration. Can you imagine if the architects designing a sky-scraper each had their own ways of creating and documenting the building's plans? The plumbing, wiring, heating systems, floors, walls, etc., would never come together. Most computer systems interface with other computer systems. The degree of system integration is rapidly increasing. The descriptions of functions and data at the interfaces must be produced and documented in a consistent manner, in much the same way as architects must have a standard way of documenting plans. This is provided by the SDLC.

You would not dream of trying to process phone bills or checks without a system due to the complexity of the process, the need for specialization, the need for continuity and the need to interface to the rest of the world. We have seen here that the process of developing systems has the same characteristics. Therefore, we need a system to build computer systems - the SDLC.

Chapter 4 SDLC Objectives

In the prior chapter a case was made for the necessity of having an SDLC and that the SDLC is itself a system. When we plan to develop, acquire or revise a system we must be absolutely clear on the objectives of that system. The objectives must be stated in terms of the expected benefits that the business expects from investing in that system. The objectives define the expected return on investment.

An SDLC has three primary business objectives:

- Ensure the delivery of high quality systems;

- Provide strong management controls;

- Maximize productivity.

In other words, the SDLC should ensure that we can produce more function, with higher quality, in less time, with less resources and in a predictable manner.

4.1 Ensure High Quality

Judging the quality of a wine or a meal is a subjective process. The results of the evaluation reflect the tastes and opinions of the taster. But we need a more rigorous, objective approach to evaluating the quality of systems. Therefore, before we can ensure that a system has high quality, we must know what quality is in a business context.

The primary definition of quality in a business context is the return on investment (ROI) achieved by the system. The business could have taken the money spent on developing and running the system and spent it on advertising, product development, staff raises or many other things. However, someone made a decision that if that money was spent on the system it would provide the best return or at least a return justifying spending the money on it.

This ROI can be the result of such things as: operational cost savings or cost avoidance; improved product flexibility resulting in a larger market share; and/or improved decision support for strategic, tactical and operational planning. In each case the ROI should be expressed quantitatively, not qualitatively. Qualitative objectives are almost always poorly defined reflections of incompletely analyzed quantitative benefits.

The SDLC must ensure that these objectives are well defined for each project and used as the primary measure of success for the project and system.

The business objectives provide the contextual definition of quality. There is also an intrinsic definition of quality. This definition of quality centers on the characteristics of the system itself: is it zero defect, is it well-structured, it is well-documented, is it

functionally robust, etc. The characteristics are obviously directly linked to the system's ability to provide the best possible ROI. Therefore, the SDLC must ensure that these qualities are built into the system. However, how far you go in achieving intrinsic quality is tempered by the need to keep contextual quality (i.e., ROI) the number one priority. At times there are trade-offs to be made between the two. Within the constraints of the business objectives, the SDLC must ensure that the system has a high degree of intrinsic quality.

4.2 Provide Strong Management Control

The essence of strong management controls is predictability and feedback. Projects may last for many months or even years. Predictability is provided by being able to accurately estimate, as early as possible, how long a project will take, how many resources it will require and how much it will cost. This information is key to determining if the ROI will be achieved in a timely manner or at all. The SDLC must ensure that such planning estimates can be put together before there have been any significant expenditures of resources, time and money on the project. The feedback process tells us how well we are doing in meeting the plan and the project's objectives. If we are on target, we need that verified. If there are exceptions, these must be detected as early as possible so that corrective actions can be taken in a timely manner. The SDLC must ensure that management has timely, complete and accurate information on the status of the project and the system throughout the development process.

4.3 Maximize Productivity

There are two basic definitions of productivity. One centers on what you are building; the other is from the perspective of how many resources, how much time and how much money it takes to build it. The first definition of productivity is based on the return on investment (ROI) concept. What value is there in doing the wrong system twice as fast? It would be like taking a trip to the wrong place in a plane that was twice as fast. You might have been able to simply walk to the correct destination. Therefore, the best way to measure a project team's or system department's productivity is to measure the net ROI of their efforts. The SDLC must not just ensure that the expected ROI for each project is well defined. It must ensure that the projects being done are those with the maximum possible ROI opportunities of all of the potential projects.

Even if every project in the queue has significant ROI benefits associated with it, there is a practical limit to how large and how fast the systems organization can grow. We need to make the available staff as productive as possible with regard to the time, money and resources required to deliver a given amount of function. The first issue we face is the degree to which the development process is labor intensive. Part of the solution lies in automation. The SDLC must be designed in such a way as to take maximum advantage of the computer assisted software engineering (CASE) tools.

The complexity of the systems and the technology they use has required increased specialization. These specialized skills are often scarce. The SDLC must delineate the tasks and deliverables in such a way as to ensure that specialized resources can be brought to bear on the project in the most effective and efficient way possible.

One of the major wastes of resources on a project is having to do things over. Scrap and rework occurs due to such things as errors and changes in scope. The SDLC must ensure that scrap and rework is minimized. Another activity that results in non-productive effort is the start-up time for new resources being added to the project. The SDLC must ensure that start-up time is minimized in any way possible. A final opportunity area for productivity improvements is the use of off-the-shelf components. Many applications contain functions identical to those in other applications. The SDLC should ensure that if useful components already exist, they can be re-used in many applications.

What we have identified so far are the primary business objectives of the SDLC and the areas of opportunity we should focus on in meeting these objectives. What we must now do is translate these objectives into a set of requirements and design points for the SDLC.

Chapter 5 SDLC Requirements

The requirements for the SDLC fall into five major categories:

- Scope

- Technical Activities

- Management Activities

- Usability

- Installation Guidance

The scoping requirements bound what types of systems and projects are supported by the SDLC. The technical and management activities define the types of tasks and deliverables to be considered in the project. The usability requirements address the various ways in which the SDLC will be used by the team members and what must be considered in making the SDLC easy to use in all cases. The installation requirements address the needs associated with phasing the SDLC into use, possibly piece by piece, over time.

5.1 Scope Requirements

The SDLC must be able to support various project types, project sizes and system types.

5.1.1 Project Types

There are five project types that the SDLC must support:

- New Development

- Rewrites of Existing Systems

- Maintenance

- Package Selection

- System Conversions

New Development

A totally new system development effort implies that there is no existing system. You have a blank sheet of paper and total latitude in defining its requirements and design. In reality this is a rather rare occurance.

Rewrites

In a rewrite there is an existing system but the current design has degenerated to become so poorly structured that it is difficult to maintain or add any significant new features. Therefore, a new system will be created to take its place. However, there is a necessity to retain a high degree of functional compatibility with the existing system. Thus, you might go from a batch system to an on-line system, from a centralized system to a distributed system, etc., but the core business (i.e., logical) functions remain the same.

Maintenance

Here we must be careful to make the distinction between a management definition of maintenance and a technical definition. From a management perspective, some organizations call any project of under six person months, or some similar resource limit, that affects an existing system a maintenance project. Some even reduce this to just the effort required to fix errors and comply with regulatory changes. This can also be called "zero-based maintenance", after zero-based budgeting, since anything over that is discretionary. The rest is called development.

We prefer to use a technical definition of maintenance to mean any incremental improvements to an existing system, regardless of the size of the changes. The rationale for this is that the techniques and tools required to go into an existing system differ from those where there is a blank sheet of paper.

Package Selection

Package selection involves evaluating, acquiring, tailoring and installing third party software.

System Conversions

A system conversion involves translating a system to run in a new environment. This includes conversions to a new language, a new operating system, a new computer, new disk drives, a new DBMS, etc. In doing the translation, the system is not redesigned. It is ported over to the new environment on a one-to-one basis to the extent possible.

In reality, projects are often a blend of these various project types. For example, a package installation may also require maintenance changes to interfacing systems,

developing some new code, converting other code to run on a compatible configuration and rewriting portions of some systems. The SDLC must handle each project type and any blend of them.

5.1.2 Project Sizes

Projects come in many sizes. Some may last as short as a day, staffed by only one person. Others may last many years, staffed by hundreds of people scattered across many development locations. The types and degree of the management controls, such as project check-points and status reporting, change depending on the size of the effort. The SDLC must accommodate the full range of project sizes without burdening the small project nor over simplifying to the detriment of the large project.

5.1.3 System Types

The SDLC must be able to support:

- Batch systems, on-line systems, real time systems.

- Mainframe, client-server, web, PC, imbedded systems.

- Centralized systems and distributed systems.

- Stand-alone systems and integrated systems.

- Automated systems and manual systems.

The SDLC must support each of these and any combination of them.

The SDLC actually needs to support the full range of combinations and permutations of the various project types, project sizes and system types. It must do this in a single lifecycle. Creating a unique lifecycle for each possible combination of the above would result in literally billions of SDLC's. (We leave the computation up to the reader.)

5.2 Technical Activities

The technical activities fall into a number of major categories:

- System Definition (Analysis, Design, Coding)

- Testing

- System Installation (e.g., Data Conversion, Training)

- Production Support (e.g., Problem Management)

- Evaluating Alternatives

- Defining Releases

- Reconciling Information Across Multiple Phases

- Reconciling To A Global View

- Defining The Project's Technical Strategy)

In addressing each of these topics we will need to distinguish what tasks must be performed from how they might be performed. This distinction is important since the "how-to" is dependent on the specific software engineering techniques selected and the available CASE tools. However, the "what" should be generic and stable, regardless of the techniques and tools.

5.2.1 System Definition

In defining the requirements for supporting analysis, design and coding we must consider three aspects of the problem: system components, the categories of requirements and system views.

System Components

Regardless of the techniques being used, we can say that any system can be said to be composed of nine basic component types:

- Use Cases

- Functions

- Triggers

- Data Stores

- Data Flows

- Data Elements

- Processors

- Data Storage

- Data Connections

- Actors/External Entities

Use Cases are an ordered set of processes, initiated by a specific trigger (e.g., transaction, end of day), which accomplish a meaningful unit of work from the perspective of the user.

Functions are context independent processes that transform data and/or determine the state of entities.

Triggers are the events that intiate Use Cases. There are three types of triggers: time triggers, state triggers and transaction triggers.

Data stores are data at rest. Data flows are data in movement between two processes, a process and a data store, etc.

Data elements are the atomic units within data flows and data stores.

Processors are the components which execute the processes and events (i.e., computers and people).

Data storage is the repository in which the data stores reside (e.g., disks, tapes, filing cabinets).

Data connections are the pipelines through which the data flows flow (e.g., communications network, the mail).

Actors/External entities are people or systems outside the scope of the system under investigation but with which it must interface.

Each of these components has many properties or attributes which are needed to fully describe them. For example, in describing a process we can state its algorithm, who or what executes it, where it takes place, when it takes place, how much information it must process, etc. Figure 1 lists the properties which can be described for the various component types.

In a given project and for a given component, the properties which must be gathered/defined may vary. The SDLC must allow for this flexibility versus an all-or-nothing approach.

Categories Of Requirements

There are at least ten major categories of requirements which must be addressed by the analysis and design process:

- Function

- Performance

- Usability

- Reliability, Availability, Serviceability

- Localization

- Portability

- Maintainability

- Security

- Testability

- Extendibility

Functional requirements define what data the system must produce and what transformations of that data the system must be able to do.

Performance requirements describe how fast it must do the transformations, how many it must do and any limitations on the amount of utilization of the agents used to support the transformation (e.g., amount of machine time, amount of disk space).

Usability requirements describe the ergonomics of the system (e.g., ease of correctly interpreting the information on a screen).

Reliability defines the degree of accuracy required in the transforms. In billing this would be 100%. In weather forecasting it could be plus or minus 5% for a short term forecast. Availability defines the amount of time the system is actually up during the time periods it is supposed to be up. This is usually defined as a percentage, qualified with standard deviations. Mean time to failure, by type of failure, further defines system availability. Serviceability addresses how quickly the system can be corrected when it is

discovered to be unreliable or unavailable. This might be expressed as the mean time to fix. Mean time to fix is usually qualified by the type and severity of the failures.

COMPONENT	FUNCTION	DATA STORE	DATA FLOW	DATA ELEMENT	USE CASE	EXTERNAL ENTITY/ ACTOR
PROPERTY						
Identification						
Name	X	X	X	X	X	X
Date	X	X	X	X	X	X
Version Number	X	X	X	X	X	X
Aliasing	X	X	X	X	X	X
Description	X	X	X	X	X	X
Cross-Reference						
Parents	X	X	X	X	X	X
Migration	X	X	X	X	X	X
Source/Sinks			X			
Inputs/Outputs	X	X			X	X
Model	X	X	X	X	X	X
Information Sources	X	X	X	X	X	X
Design Axioms	X	X	X		X	
Component Composition						
Census	X	X	X		X	
Organization		X	X			
Syntax/Structure		X	X			
Trigger					X	
Processes					X	
Process Parameters	X				X	
Algorithim	X					
Processing Characteristics						
Rate of Flow			X			
Volume	X	X				X
Access		X				
Periodicity	X					X
Rate of Processing	X					X
Processing Windows	X					X
Maintenance		X				
Security	X	X	X		X	X
Integrity (R.A.S.)	X	X	X		X	X
Priority	X				X	X

FIGURE 1

COMPONENT	FUNCTION	DATA STORE	DATA FLOW	DATA ELEMENT	USE CASE	EXTERNAL ENTITY/ ACTOR
Agent Chartacteristics						
Data Connection Char.			X			
Medium			X			
Size			X			
Encoding Method			X			
Connection Requirement			X			
Type			X			
I.D.			X			
Source/Sink			X			
Flow Requirements			X			
% Capacity			X			
R.A.S.			X			
Environment			X			
Data Storage Char.						
Medium		X				
Size		X				
Encoding		X				
Storage		X				
Type		X				
I.D.		X				
Location		X				
Volume		X				
% Capacity		X				
R.A.S.		X				
Environment		X				
Processor Characteristics						
Type	X					
I.D.	X					
Location	X					
Size	X					
% Utilization	X					
Required Features	X					
R.A.S.	X					
Environment	X					
Costs						
Cost per Unit	X	X	X		X	X
Cost for Availability	X	X	X			X
Value						
Replacement Cost	X	X	X			X
Fair Market Value	X	X			X	
Business Value	X	X	X		X	

FIGURE 1 CONTINUED

Localization describes the ability to adapt the application to different languages, character sets, and cultures to support international users.

Portability describes the need to be able to quickly adapt the application to run on different technology.

Maintainability describes the need for people to be able to quickly and reliably identify where changes must be made to the system.

Security describes the systems ability to restrict who can do what to what and when.

Extendibility describes the system's ability to absorb major modifications to changes in any of the above requirements, while minimizing the impact to the rest of the system. This is usually described in terms of change scenarios accompanied with the probability the change will be needed and the probable time frame in which it will occur.

Depending on the application, there may be some other "ilities" that are important. For example, systems supporting people who travel a lot, such as salesmen, have a requirement for ease of physical portability. However, in general the above six categories are sufficient.

System Views

The analysis and design approach must allow the system definition to be viewed from an external viewpoint and an internal viewpoint. The external viewpoint is also often referred to as the logical or user or requirements-oriented view. The internal viewpoint is also called the physical or design oriented viewpoint.

An SDLC must also allow these various views from the perspective of the total system, from a single component, or anything in between.

One problem we face in this discussion is terminology. In evaluating an SDLC the important thing is whether or not each item is fully addressed. It is less important what these things are called or how the collection of them is organized.

Analysis And Design Techniques And Tools

In the above discussion we focused exclusively on what information must be gathered in the analysis process or defined in the design process. This information is totally independent of how we gather or define it. The SDLC should, however, also aid us in showing us how the various analysis and design techniques and tools fit into actually doing this work. Unfortunately, we have a cornucopia of techniques to choose from. Do we use the process driven approach of Yourdon-Constantine or the data driven approach of Jackson-Warnier-Orr? Do we use JAD's? Do we use rapid prototyping?

The list of the techniques could go on and on. The optimal approach would be for the SDLC to describe how each of the various techniques - the "how to" - can be used to create the above information - the "whats". This would leave the "how to" up the project teams. This flexibility is desired since the "best" technique is dependent on the application type, available tools, and personal preferences and experiences.

5.2.2 Testing

In defining the SDLC's testing oriented requirements, we must break this topic up a bit. We need to discuss what is being tested, when testing must occur, the steps in testing, the properties to test for and the definition of the overall testing effort.

What To Test

What do you test? In a word - everything. The objectives must be verified. The requirements, design and code must be tested. The user manuals must be tested. The installation support systems and materials must be tested - e.g., training materials, data conversion programs. Even the test support system must be tested. The SDLC must ensure that all products are tested in a rigorous manner.

When To Test

Numerous studies have shown that the earlier you test, the cheaper the costs. The cost difference between finding an error early in the requirements definition versus once it is in production have been found to be as much as 270 to 1500 times as expensive. Prior to system release the issue is scrap and rework. The later an error is found, the greater the impact on work already completed. It is like finding out after a building is constructed that the plumbing has been omitted. If someone had noticed this omission early in the blue-print stage, the costs would have been minimal.

After system release, the issue is not just scrap and rework, but the impact that the error had on the business and its customers. There are hundreds of examples of a simple error, from the perspective of what it took to fix the code, costing the business many millions of dollars.

The issue of when to test is tightly coupled to the issue of knowing when a task is complete. Each task should result in a specific set of outputs. The task cannot be considered complete until these outputs have been verified to be complete and correct. This is done via testing. Therefore, testing is a part of every step, not just an activity done once code is written or just at phase checkpoints. The SDLC must integrate testing throughout the entire project lifecycle so that each work product is tested as it is created.

Testing Steps

Regardless of the specific techniques and tools used for testing, there are six basic steps to testing anything. These steps are:

- Define Test Criteria

- Design Test Cases

- Build Test Cases

- Execute Tests

- Verify Test Results

- Store Test Cases

The first step is to define the test criteria. The criteria defines how to measure the success of the testing and defines what is considered the completeness criteria for testing. In other words, it tells you when you are done testing and whether or not the product passed the test. An example of this is to set a criteria that testing must execute every functional variation, every code statement and every branch vector at least once.

The second step is to design tests that meet the desired success criteria. The techniques employed should be rigorous and algorithmic rather than subjective.

The third step is to build the test cases. This process could range from creating paper based tests used in walk-throughs to building full test data bases and test transaction files used in machine-based testing.

The fourth step is to execute the tests. This process is performed both by computers and people. For example, this is where the walk-through process fits into the overall testing activity. It is the test execution process used before the code is built. It does not, in itself, define the criteria, design the tests or build them.

The fifth step is verifying the results of the test executions. You must ensure that everything was tested (based on the coverage criteria) and that you got the correct answer for every test (based on the test case design).

The sixth step in testing is storing the tests in such a way that they can be used in future testing efforts, both within this project and for future releases.

The SDLC must provide guidance for doing each of these testing steps.

Properties To Test For

In our discussion of the categories of requirements we identified function, performance, usability, reliability, availability, serviceability, localization, portability, maintainability, security, and extendibility. The testing process in the SDLC must test for each of the properties.

Definition Of The Overall Testing Effort

Summarizing the above discussion, the SDLC supports testing every product, at each stage in its creation, for each of the desired categories of requirements/design characteristics, via the six basic steps of testing.

In defining the specific techniques and tools to do this the process gets even more complicated. The reason is that many of the testing steps can be affected by the architecture of the system and its target configuration environment. For example, you may need a different test coverage monitor for each language the code is written in, each operating system the code will run on and each machine the code will run on.

Since the testing process is such a key aspect of the overall development process, the SDLC must provide information and guidance on what and how each of the testing concerns will be addressed.

Note: In defining the overall testing process for a given project, many of the issues are the same as for the installation activities and will be covered with them. (See "Defining The Installation Process" in the next section.)

5.2.3 System Installation

There are five topics which describe the process of converting from the existing system to the new one:

- Data Conversion

- Training

- Facilities Installation

- Hardware/Software Installation

- Production Cutover

Data conversion is the process of translating the data from the existing data formats and data definitions to those required by the new system.

Training is the process of teaching all users of the system how to use the functions and features of the new systems. Users may include operational staff, management, computer

operations, maintenance programmers and even customers. The training process must factor in cutover training and on-going training. Cutover training focuses on mapping the old way of doing things onto the new way. On-going training addresses the fact that, as turnover occurs, new people will have to be fully trained from scratch on the production system.

Facilities installation addresses the installation and set-up of the office space, phones, desks, etc., required to support the manual portions of the system. It also covers the physical preparation of the environment in which the computers will be installed such as raised floor, power, cooling, etc.

Hardware/software installation is the process of installing the vendor supplied computer configuration and supplemental applications required to support the system being developed. This includes both the development/test configuration and the production configuration.

Production cutover is the process of defining and supporting the approach for phasing the new system into production. For example, you could do a drop dead cutover where everyone is cut over to the new system at the same time. You could also phase the system in one site at a time. You could even phase it in function by function. (Note that a phased installation of function is distinct from phased development of those functions.)

Defining The Installation Process

For each of the installation topics, and for the overall testing effort as well, there are a number of activities which must take place:

- Setting Goals

-Doing An Impact Assessment

- Defining The Strategy

- Defining The Stages

- Defining And Building The Support Systems

Setting Goals:

The project has a set of business goals that define the expected ROI for the project. Some of these will be reflected in the specific installation topics. For example, a project goal to improve customer service could be reflected in a data conversion goal to scrub the existing files for redundant or inaccurate data as part of the data conversion process, thus improving operational efficiency. That same project goal might manifest itself in production cutover as a goal to phase the new system in with minimal disruptions to customer service. Each project goal, then, must be reviewed from the perspective of each

installation activity to define goals for that activity which are supportive of the overall project goals.

Impact Assessment:

Each installation/test activity can be complex and expensive in its own right. How rigorous to be is dependent on the degree of change represented by the new system and potential problems that might be caused if the activity is not handled perfectly. For example, a system that monitors patient care in the critical ward of a hospital could cause a death if there are any errors in it. On the other hand, a one time report on the number of patients treated this year may have little impact if slightly off. The impact of an error would be very different and thus the necessary level of testing would be different.

Strategy:

For each of these activities, the basic approach for doing them must be defined. For example, data conversion could be done using separate data conversion programs or via the production data base build transactions. The choice of the approach will affect the schedules for the completion of the build functions, bound the required support systems and affect the procedures for ensuring the integrity of the new data bases.

Stages:

Any of the installation activities can be, and usually are, done in stages. The testing of code may go through unit, component and system testing. Training might be staged to first train the trainers, then a pilot group, then the rest of the organization. For each stage you need to define:

The units (e.g., specific people to train in a group; specific modules to test together

The sequence of the units (e.g., train department A, then department B)

The support system dependencies (i.e., the portions of the supporting programs, tools and procedures needed for this stage)

The entry/exit criteria for the stage (i.e., the quantitative and qualitative criteria for beginning and ending each stage))

Support System:

The support system includes the systems and procedures that will be used to perform the activity. This might involve developing the support system (e.g., data conversion

programs), acquiring an existing tool from a vendor (e.g., a test monitor), and/or simply using pre-existing components from a prior effort.

The SDLC must contain the steps and guidelines for defining each of these aspects of the various installation/test efforts.

Integrating The Installation And Test Activities

There is a requirement to see if the needs of one activity depend on the results of another or if the results of one activity could be useful to another. This allows optimization of the installation activities from the project perspective. For example, the initial user training effort could be synchronized with the testing effort to allow the users to aid in the execution of the test scripts. Similarly, the test cases, which should represent one of everything the system does, could be used as materials for the training classes. Thus, the SDLC should contain the activities to reconcile and synchronize the installation/test activities with each other in a timely manner.

5.2.4 Production Support

Once the system is put into production, there are a number of on-going activities that directly support it. These are:

- Problem Management

- Release Management

- Publications Support

Problem management addresses the issue of how errors in the system encountered by the users/customers in production will be handled. This includes error reporting procedures (e.g., production hot-lines), problem diagnostics, error correction procedures and problem tracking (e.g., mean-time-to-failure, mean-time-to-fix by problem severity).

Release management addresses the issue of how new versions of the application will be distributed and how version control will be managed when different sites have different versions of the system.

Publications support concerns the logistics problems of physically producing and distributing the various user manuals.

These issues have traditionally only been of concern to the system vendors. For example, IBM is the second largest publisher in the world (the U.S. federal government is the largest). However, as applications have grown larger, more complex and distributed, the problems in-house applications groups face take on more of the characteristics of a vendor supplied product. The SDLC must include the activities to analyze these needs and prepare the appropriate support systems.

For each of these activities there are a number of items which must be considered:

- Goals

- User Profile

- Strategy

Support Systems

The definition of support goals is the same approach taken in defining the installation goals - i.e., what portions of the overall project and system goals will be specifically supported by these activities and how should they be expressed from that activity's perspective.

The user profile analysis addresses tailoring the support processes to how the production system is used. The problem management needs would be different for an on-line system up 24 hours a day 7 days a week than for a batch system run between 3PM and 5PM Mondays.

The support systems are the computer systems and manual procedures used to support the activity.

The SDLC must ensure that each of these topics is addressed in a thorough and timely manner.

5.2.5 Defining Releases

Attempting to build and install a very large system in one massive project almost guarantees failure. The larger the project scope the more time, people, coordination, etc., are required. At some point the size reaches the point where no one can get their arms around it and control it. The long time frames needlessly delay many benefits which could be achieved with a small subset of the functions (the 80/20 rule). The degree of change on the user that the total system represents can be overwhelming. Even if you could build the system, the user could not absorb it into their operations.

For all of these reasons and many more, it is often advisable to break a given system into multiple releases. In doing this you must ensure that:

- each release meets specific business goals;

- that each release delivers a coherent set of functions;

- that the releases are staged and designed in such a way as to minimize scrap and rework in the subsequent releases;

- that the installation support and production support are also properly staged in concert with the application releases.

The SDLC must ensure that the process of defining releases is robust enough to cover all of these issues.

5.2.6 Evaluating Alternatives

For a given set of objectives there may be many ways to meet them. It is not unreasonable to explore in some level of detail more than one alternative before deciding on the final approach. This can actually become a complex task. Consider that for a given set of objectives there are one or more scopes of analysis. For each scope of analysis, there are one or more possible designs. For each design, there are one or more installation approaches and one or more production support approaches. The fan out of alternatives grows exponentially. To make it even worse, there are, for each one of these alternatives, one or more options in staging them in releases.

Even in evaluating a given alternative, there are various levels of detail to which you can go before making your decision. In large complex systems where the stakes are very high to the business, some organizations have chosen to actually build and install in pilot programs more than one alternative before making the final decision. The SDLC must support the creation and evaluation of alternatives to whatever degree the business decides is appropriate for that system.

5.2.7 Reconciling Information Across Phases

All SDLCs break the project lifecycle into phases. This is one of the most basic management controls. One implication of this is that the processes of analysis, design, installation planning, production support planning and testing can be spread across multiple phases. The SDLC must ensure that the work has continuity from phase to phase.

5.2.8 Reconciling To A Global View

Very few systems and projects exist in isolation from other systems. There are many levels of integration. A low level of integration occurs where multiple applications share a given hardware/software configuration. It must be validated that the next application will not degrade the execution of the others. Another example of low level integration for systems is the installation's standards such as program naming conventions.

At a higher and more tightly coupled level there is the creation of truly integrated systems. Systems tightly integrate in a number of ways: via shared data bases, via shared networks, via shared interface standards (e.g., screen formats), via shared application utility functions and via shared off-the-shelf code. When this level of integration is desired, then there must be an Applications Architecture group(s). This group has the responsibility to define the global structure and to ensure that each team is conforming to it.

The process of reconciling what one project team is doing to this global view is not something that just happens at a few major checkpoints. It must be an on-going activity. The SDLC must address how best to perform this activity in a complete and timely manner.

5.2.9 Defining The Project's Technical Strategy

For every technical topic that we have just covered there are numerous decisions to be made. We need to decide such things as what information is needed, how best to gather or create that information, how to organize and display that information, and how to store and maintain the information. This involves many decisions about what tasks to do (e.g., do we need worry about hardware/software installation for this project). It also involves decisions about what software engineering techniques to use and which CASE tools to use.

These decisions are not just made once at the beginning of the project. The best technical strategy for getting started may not be the best one for detailing out the system definition. Thus, the SDLC must make this an explicit set of tasks at appropriate points throughout the lifecycle.

The change control procedure contains a number of key steps. The first is to determine who will decide the fate of a requested change. The second is to define the procedures for submitting and evaluating changes. The third is to identify the points in the SDLC process where each given deliverable will come under change control. The SDLC must describe how each of these steps will be accomplished.

5.3 Management Activities

Project management is the process of planning, organizing, staffing, directing, coordinating and controlling a project. These activities are as much a part of the SDLC as the technical tasks identified in the prior section. Specifically, the SDLC must contain tasks and guidance on the following topics:

- Setting Priorities

- Defining Objectives

- Planning And Estimating

- Tracking/Status Reporting

- Change Control

- Step Wise Commitment

- Cost/Benefit Analysis

- Risk Assessments

- Interacting With Users

- Managing Vendors

- Post Implementation Reviews

- Quality Assurance Reviews

5.3.1 Setting Priorities

There are two levels of setting priorities. The highest level concerns managing the project queues for the entire systems department. The second level of priority setting occurs within a system and/or project.

Prioritizing The Department Project Queues

The overall project queue is, or should be, the result of the strategic systems planning process. In this process, the strategic business plan is analyzed to determine what system support is required to support the organization's initiatives. This is actually a two way process since technology creates business opportunities.

The strategic systems planning process is generally viewed as outside of the scope of an SDLC. The SDLC begins with a specific project request or idea. However, the SDLC must interface to this process. The priorities in the overall queue are based on a number of assumptions about the potential ROI and timing of the various projects. The SDLC must support being able to quickly verify or modify these assumptions. This information is then fed back to the systems planning process so that adjustments to the priorities can be made.

Similarly, the SDLC interfaces with the product planning process. The analysis of the strategic business plan may result in a number of product planning projects. These in turn may result in systems projects.

In essence, the strategic business planning process forms the highest level context. It then spawns and bounds the strategic systems planning process and product planning process. These in turn drive the creation of the project queues which trigger initiating the activities in the SDLC. The SDLC, therefore must be able to interface to these other, more global, planning activities.

Prioritizing Within A System/Project

For systems under maintenance, resources are not allocated at the beginning of the year to specific changes. Resources are allocated on a level of effort basis - e.g., system X will be allocated ten person years worth of support this year. How those resources are specifically allocated will be decided as part of an on-going process between the user and the systems support team.

Similarly, this same process can occur even within a development effort. There are always trade-offs to be made between time, function and resources in meeting the project's objectives. This affects the scope of analysis and the functions to be included within the current release. Again, priorities are being set as to what are the most important things to do.

The SDLC must support the managing of both the maintenance priorities and the intra-project development priorities.

5.3.2 Defining Objectives

Objectives

Objectives are the statements of what benefits the organization hopes to achieve as a result of investing in the project and the system. They define **why** the project is being done. It is a distinct statement from **what** is to be delivered.

A fully stated objective defines the desired ROI in quantitative terms. Qualitative benefits are usually poorly defined quantitative benefits. The objectives are further refined by stating the time frames and resource limits within which the objectives are to

be met. The SDLC must ensure that fully defined objectives are identified at the beginning of any project. It should then use these objectives to guide the process of analysis of design to ensure that the project stays focused on these objectives. The SDLC should use these objectives as the primary measure of the success of the project and the system.

This statement of goals is distinct from the cost/benefit analysis. The objectives define the **desired** ROI. The cost/benefit analysis defines the **projected** ROI based on what is known about the project and the system under development. These may exceed or fall short of the desired ROI. The post implementation review identifies the **actual** ROI achieved by the project and system. The SDLC must maintain the distinction among objectives, cost/benefits and post implementation analysis.

System Objectives Versus Project Objectives

When a system is first created, it is created with specific on-going objectives in mind - e.g., reduce operational costs. Thereafter, numerous projects will modify that system. Each project will have its own distinct objectives. However, the cumulative effect of these incremental projects might negate the system's ability to meet its overall objectives. The SDLC must make the distinction between system objectives and project objectives. It must also include the activities for reconciling these two and updating the system objectives from those project objectives which reflect on-going goals.

Reconciling Objectives

Individual projects often affect many different user groups. Each group may have their own idea as to the project's objectives and the priorities of those objectives. For example, the operations group would be concerned about operational efficiency while the marketing group is concerned about market share. The first might imply a functionally simple version of the system while the second might want all the bells and whistles possible. Somehow these apparently conflicting objectives must be reconciled and prioritized. The SDLC must make this process explicit and provide guidance on how to accomplish it.

5.3.3 Planning And Estimating

A project plan is composed of tasks, dependencies, resources, schedules and budgets. The tasks define what you will do and what will be produced. The dependencies determine the order in which the tasks must be done. The resources identify who will do the tasks. The schedule states when the tasks will begin and end. The budget defines how much all this will cost.

Generic Project Plan

The cornerstone of the planning process is the completeness of the task list. The are two sources of information for identifying the required tasks. The first is the SDLC itself. It is the generic project plan containing the master check-list all of the tasks and deliverables which the project manager must consider.

The SDLC must be robust enough to provide this complete generic list regardless of the project's scope, etc. It should identify the dependencies between these tasks. It should provide guidance on the degree to which these tasks can be overlapped. This would allow the project manager to optimize the plan for minimizing resources or for delivering the product in the shortest amount of time.

Specific Project Plan

Most of the specific tasks come from the results of the analysis and design effort itself. For example, the SDLC would identify at what point the data base design effort is done and would describe the tasks for doing it. However, it is the results of the project's analysis that would tell how many data bases must be designed.

This presents us with a problem. How can we plan the project if the plan is the result of outputs from the project? The basic issue is that projects are research and development efforts. Until the research portion of the project is completed, you do not know how long it will take, how much it will cost or whether the system can even be built. This is because there are still unknowns which can affect the project's scope and feasibility.

The SDLC must recognize the research and development nature of projects. It must ensure that the research portion of the project is completely and thoroughly done as quickly as possible. It must also recognize that planning is an iterative, on-going process throughout the entire project.

Estimating

A plan is basically a prediction of the future - an educated guess. It contains many estimates. You estimate how large the deliverable from each task will be. You estimate the productivity rate that will be achieved in producing that deliverable. You estimate the elapsed time involved for the task. The accuracy of these estimates will vary depending on the amount of information available, who is doing the estimates and the techniques used.

The completeness of the information available to the planning process was addressed above. The SDLC can do nothing about who is doing the estimating. However, the SDLC must support the estimating process in a number of ways.

First, the SDLC should discuss the various formalized estimating techniques that have been developed over the years (e.g., expert judgment, analogy). Second, the SDLC

should support the accumulation and use of information from past projects. The SDLC should include the activities for using and updating this project history data base.

Reconciling To A Global Plan

Projects today require many specialized skills. These skills are often collected into groups which support many projects at the same time (e.g., data administration, test teams). The quantity and timing of these shared resources must be synchronized with all the other projects that need those same resources.

Planning conflicts can arise over the facilities that a team needs to complete its work. For example, if too many projects enter system test at the same time, then there might be a problem with computer time and disk storage. The facilities needs must be reconciled across the various projects.

Planning conflicts can also arise over the areas affected by the projects. If too many projects are all affecting the same user area, the degree of disruption could be too great. The timing of delivering these new systems must be coordinated so as not to put the business at risk.

In doing integrated systems, the creation of a given application may depend on the results of a number of project teams all delivering their systems at the right time. Each project's plans must be synchronized with all the others.

The SDLC must, therefore, contain the tasks for synchronizing a project's plan to the other plans which are linked to it.

5.3.4 Tracking And Status Reporting

The plan is a statement of what we guess will happen. As the project progresses we need to keep track of how closely our guesses were to reality and report this to the appropriate people. This is accomplished via the tracking and reporting activities.

Tracking

In tracking the project we are continually comparing the plan with our actual results. This requires that what we are tracking have well defined completion criteria. Otherwise, how will we know if the task is done or not? The SDLC must ensure that the completion criteria for each generic task is objectively, not subjectively, defined.

Status Reporting

There are two types of status reporting: periodic and checkpoint. Periodic status reports are prepared on a weekly, monthly, etc., basis so that management can keep track of how the project is doing against the plan. These status reports are also a means for the project manager to communicate requests for assistance in resolving issues.

Checkpoint reporting requires the project manager to summarize the overall status of the project and system for management when certain checkpoints have been reached. These usually are defined by the SDLC's phases. The objective is to bring management up to date as well as provide the opportunity for reassessing whether to continue or not. The SDLC must support both types of status reporting, defining the guidelines and contents for preparing them.

Feedback Reports

There is another type of report which is aimed at providing feedback on specific activities. For example, the training effort may have been planned in stages, using various training materials and methods. After the initial round of training, there should be a report which summarizes how well it went. This provides feedback to the next stages of training and to future projects as well.

The SDLC must include the activities and report outlines for preparing these feedback reports. These include test reports, data conversion reports, training reports, facilities installation reports, hardware/software installation reports, production cutover reports, quality assurance review reports and post-implementation review reports.

5.3.5 Change Control

Projects can be described as a process containing four interdependent variables: time, function, resources and quality. For a given amount of function, there is some minimum amount of time and resources required to create a quality product. If you change the functions, you must change time and/or resources. Otherwise, you reduce quality and/or reduce the probability that you can even deliver the functions at all within the time and resource constraints.

When the project was initiated, there were certain assumptions about its scope. The scope was further delineated by the initial analysis and design activities used to determine feasibility. If the user wants to change that scope they are basically asking to renegotiate the "contract" that it has with the project team.

Change is also disruptive. The later that a change is requested, the more scrap and rework that is required. However, since the user cannot "freeze" their environment, the project team cannot "freeze" the system definition. What is required is a formal change control process.

Identifying Changes

The first issue in establishing formal change control is delineating a change from a clarification. The user will often profess that what they are asking for is not a change at all. It was what they thought they were getting all along. The SDLC must ensure that the inventory of what is to be delivered is quickly and concisely bound.

Change Control Procedure

The change control procedure contains a number of key steps. The first is to determine who will decide the fate of a requested change. The second is to define the procedures for submitting and evaluating changes. The third is to identify the points in the SDLC process where each given deliverable will come under change control. The SDLC must describe how each of these steps will be accomplished.

Change Control And Planning

Even if a change request is turned down, it takes resources to evaluate it. While the evaluation process may consume only a small portion of the project's resources, it tends to take up a large percentage of the time of key resources. The SDLC must ensure that the resources required to evaluate changes are included in the project plan.

If a change request is accepted, it will require resources to complete it. The SDLC should assume that some of level of change will occur and account for it in the planning process. However, it should also provide guidance on what to do when those resources have been used up and the plan must be recast.

5.3.6 Step Wise Commitment

Receiving Concurrence To Proceed

As stated before, projects are research and development efforts. Therefore, it is unreasonable to ask management to commit to the entire project from day one. The SDLC must support the process of periodic project reviews by management of whether or not it will allow the project to continue. This is an extension of what was discussed in the status reporting topic. It extends the requirements to include the specific activities to get management's concurrence to proceed.

Commitment Versus Allocation

The SDLC must differentiate between committing resources versus allocating them. When a project is authorized in the budget, the funding and people resources must be set aside or allocated for this effort. Allocating the resources places the project in the "In Plan". There should also be an "Out Plan" containing the lower priority projects to which no resources have been allocated. As a project progresses management commits, phase

by phase, to spending the allocated resources. If it is decided to terminate the project, the unused allocated resources are now available to move a project from the "Out Plan" into the "In Plan".

Psychological Commitment

There is another type of commitment to which the SDLC must be sensitive. That is the psychological commitment of the project team members. The more people that are involved in the project and the longer it goes on, the more difficult it is to stop it, even if there are periodic checkpoints for getting concurrence to proceed. This is because people like to finish what they start. It also occurs because their professional reputations are at stake.

The SDLC must do two things to minimize the negative aspects of this type of commitment. First it must support determining the project's feasibility with as small a group as possible, with a minimum amount of expenditure and in a minimum amount of time. This minimizes the psychological investment of the initial team members. Second it must support the concept that, in research and development, it is still a successful project if you found out as quickly and as cheaply as possible that you could not do it.

5.3.7 Cost/Benefit Analysis

As stated before, the cost/benefit analysis is an attempt to calculate to what degree the project and system will meet the objectives. The SDLC must address two topics in its support of this area: the scope of the analysis and the algorithm for doing it.

Cost/Benefit Scope

The benefit side of the analysis should be expressed in quantitative terms whereever possible. Qualitative or intangible benefits usually are reflections of poorly analyzed tangible benefits. The SDLC should support the process of quantifying all benefits.

On the costs side, the SDLC must address development costs, installation costs and on-going operational costs. In doing these calculations it should differentiate between capital costs and expense costs.

Cost/Benefit Algorithm

The method of calculating and displaying the cost/benefit analysis can have a profound impact on the effectiveness of making the right project commitment decisions. The calculation methods must be sophisticated enough to take into account such things as cash flow, hard dollars versus soft dollars, regional cost differences, cost trends, revenue trends, real and present value, residual values, tax implications, burdened and unburdened costs, and the impact of synergy.

The analysis is usually displayed on a spread sheet. Since spread sheets are used primarily in accounting and since accounts are supposed to be 100% accurate, the analysis appears to be more black and white than it is. An analysis of this sort is basically an educated or sophisticated guess. There is some probability that it is off by some degree. By applying such techniques as Baysian analysis, displayed on graphs, you begin to see the probabilistic nature of the process. The SDLC must provide guidance on the method of the calculation and on the means of displaying the results. Since the degree of accuracy depends on the amount of information available, the cost/benefit analysis should be an iterative process occurring at a number of points in the SDLC process.

5.3.8 Risk Assessment

Projects, being research and development efforts, entail some degree of risk. These risks affect the probability of achieving the results predicted by the cost/benefit analysis. There are business risks, technical risks and people related risks.

For each type of risk there is an active risk and a passive risk. Active risks define what might happen if you do undertake the project. Passive risks identify what might happen if you do not do it.

For each of all of these risks there are also upside risks and downside risks. Upside risks define the minimum problems you will encounter. The downside risks define the worst that could happen.

The SDLC must support the process of risk assessment including active/passive and upside/downside for each type of risk. The overall design of the SDLC should be laid out in such a way as to minimize risks where it can.

5.3.9 Interacting With Users

The user is the customer for the product being delivered by the project team. What their involvement is with the project usually defines the difference between success and failure. The SDLC must address a number of topics in this area: identifying the stake-holders; on-going user involvement; user systems owners and user project managers; and the differences between the users' environment and the project team's.

Identifying The Stake-Holders

The first issue the SDLC must address in this topic is identifying who all the users are. Any given project may find many different user areas who feel they have a stake in it. These stake-holders must be identified, their roles in the project defined and their representatives named.

On-going User Involvement

The user does not just submit a project request and then review the results at the end of the project. They must have an on-going role in the development process. The SDLC must identify each point in the process where they are involved and clearly delineate their responsibilities from those of the systems staff.

User System Owner / User Project Manager

For each system there is a specific group and a specific manager within the systems department who have responsibility for it. Unfortunately, in many organizations there is no counterpart within the user community. Since a given application may affect many user departments, the problem arises of who decides what should or should not be done to the system. This is often resolved by committee or, by default, by the systems team. There is a requirement for each system to have an identified individual from each area who has responsibility for representing that area's position. There is also a need for one of these users to be designated the primary system owner. The SDLC should explicitly support this concept.

For a given project, only a few of the areas affected by the total system may be involved. In such cases, there is a need to designate which user representative will be the user project manager. The SDLC must support this concept and differentiate it from the user system owner concept.

User Versus Systems Environment

A fundamental difference exists between most user organizations and the systems organization. Most user organizations are involved in managing a process. A process is a repetitious activity and is managed by exception. It is very predictable. The systems group does projects. Projects are research and development. By nature they are unpredictable until the research portion is done.

This difference leads to some fundamental problems. The users can quickly and accurately state what the impact of a specific request will be on their work load. They do not always understand why the systems group cannot do the same thing in estimating their project requests. Therefore, they try to control the project by prematurely fixing dates since they can read a calendar. They fix resources since they can read a spread sheet. What they have difficulty in quantifying is function and its relationship to meeting the objectives.

The result is that, for many projects, the objective becomes to deliver something, anything, by the stated date within the stated budget. Whether this meets the business objectives is immaterial. Under these circumstances, the project team takes the path of least resistance in getting the project "completed". The SDLC and the accompanying disciplines go out the window.

The SDLC must be designed in such a way as to reflect this very real issue. If it does not, even if it meets every other objective and requirement stated in this document, it will only be a pile of unused paper.

5.3.10 Managing Vendors

More and more, systems groups rely on the products and services of outside vendors in order to accomplish their goals. Products range from acquiring the hardware/software configuration the application will run on, the application itself, and the tools used to create and maintain the applications. Services range from supplemental resources to cover peak load resource demands to consultants with specialized skills. The SDLC must include the activities for ensuring the effective use of vendor supplied products and services.

Evaluating Vendors/Products

The SDLC must include the activities for first deciding whether outside products or services are even needed. If they are needed, the SDLC must provide guidance on when and how to evaluate the viability of the product/service and how to evaluate the viability of the vendor.

Pre-Contract Vendor Interaction

Many vendor situations are competitive. This requires that the process of vendor interaction, prior to selecting the final one, must be fair. Otherwise, there is a legal exposure. This is especially true where government contracts are involved as part of the overall project. The SDLC must define the process of preparing and disseminating requests for information and requests for prices.

Contract Definition

Once a vendor is selected, the details of the contract must be defined and negotiated. This is not just a task for the legal department. The legal department essentially "codes" into the contract the requirements given to it by the users and systems groups. Without these requirements the contract suffers from the same result that would happen if the systems group built a system without user requirements.

The SDLC must identify all the contract issues, in easy to understand terms, that must be considered from a business and technical perspective. It must also identify the points in the development process where the legal staff interacts with the rest of the project team.

Contract Management

Even after the contract has been signed there are issues concerning managing to the letter and spirit of the contract. You can, by your actions, inadvertently redefine the terms of the carefully defined contract.

There is also the topic of issue resolution. Resolving issues with a third party is vastly different than resolving internal issues. The SDLC must provide guidance on all aspects of contract management that the project team will have to directly contend with.

5.3.11 Post Implementation Reviews

Post implementation reviews were mentioned before in the reports section. However, this topic needs further definition in our requirements. The post implementation review provides an analysis of and feedback on the project and the system created by the project. The objective of the review is to see how well things went and to provide feedback on how to improve things the next time around.

The project was initiated on an assumption that there was going to be a specific ROI resulting from the effort. The review evaluates to what extent this was achieved. The review looks to see how well the user requirements were met by the system. It also looks at how well the team did in meeting the plan, how well the system installation effort went, how well the SDLC and tools worked, and how well the project team worked with the users and each other.

The SDLC should support a thorough post implementation review process. Since some things can best be evaluated while they are fresh in everyone's mind and others need the perspective of time, this process should be performed at intervals.

5.3.12 Quality Assurance Reviews

In developing systems, the systems are complex and the process that produces them is complex. What is needed to ensure that everything is going well and as perceived by the team is an independent expert review. This is the quality assurance review.

The objective of the Q/A review process is to ensure that quality products are produced within the project's constraints. There are two basic ways to approach this process. One is via the policeman role and the other is a more proactive role.

In the policeman approach, products are reviewed after they are completed. Unfortunately, it is then often difficult to implement any of the recommendations because the costs of rework would be prohibitive.

In a proactive role, the Q/A process begins when the approach to producing the deliverable is being decided. The Q/A staff then guides the team in the use of any of the

tools and techniques they may need but are not experienced in. While some objectivity is lost, the goal of producing quality systems is better met.

The SDLC must support a proactive and timely role for the quality assurance review process. It must identify all the points in the process where the Q/A reviews are performed and provide guidance on how to do them. It must also keep clear the distinction between the Q/A review processes and the testing processes.

5.3.13 Recurring Nature Of Management Tasks

With the exception of defining a project's objectives, each of the management topics discussed above contains recurring tasks throughout the project lifecycle. Plans must be continually updated to reflect the current understanding of the project's scope. The cost/benefit analysis and risk assessments must be revised at various points in the project. The SDLC must identify all the points in the project where it is necessary to revisit each of these topics and define exactly what must be addressed at each point.

5.4 SDLC Usability Requirements

5.4.1 Types Of Users

The SDLC is a complex system. Its users are the systems staff and anyone else who is attached to the project team for the duration of the effort. This includes user representatives, legal staff, vendor personnel, etc. The material must be organized in such a way that any given SDLC user can go directly to the pieces that concern them without wading through the entire thing.

5.4.2 Levels Of Expertise

Another usability concern that the SDLC must address is the varying levels of expertise that the users will have. Some will be expert users who only need a checklist level of detail to ensure that they do not forget anything. Others will be knowledgeable users who will want a refresher on a specific topic. Some will be experienced at doing systems, but need to understand the specifics of this SDLCs approach. Still others will be novices at doing systems in general.

For each of these cases the SDLC must ensure that the appropriate level of detail is available and that it is easy to access according to the user's needs.

5.4.3 Realistic

Perhaps the uppermost thing that will affect the acceptance of the SDLC will be to what degree it reflects a realistic way to do projects. The SDLC must not be a theoretical exercise which describes the best way to do things in an idealized environment.

Everything in the SDLC, in part and in total, should have been done on real projects with real people and real constraints. It must be a proven way of doing projects.

5.4.4 Examples Of Deliverables

One problem the SDLC must overcome is semantics. The same word may mean vastly different things to different people. "Requirements" may mean just a high level description to one person and a detailed definition to another. It may just describe function or it may also include performance, usability, etc. It may or may not include the objectives.

The only viable way around this is to include specific examples of what each output from the SDLC contains and what it looks like. Ideally this should be done three ways. First, there should be an example of each deliverable - e.g., the requirements document - along side the formal definition of its structure and contents.

Second, there should be examples that show the state of each deliverable after each process that changes it. This helps clarify exactly what each step really does.

Third, there should be a case study showing a complete set of documentation for one system and project. This shows how each of the documents relate to each other.

5.5 Installation Support Requirements

The total SDLC is a large and complex system containing many tasks, tools and techniques. It usually represents a major cultural change from what the staff have been doing in the past. It must be installed with the same kind of planning and support given to installing any complex system. That process requires training, guidance and a phased installation.

5.5.1 Installation Planning

The SDLC should provide guidance on the overall approach to installing it. This installation strategy should identify the high leverage items, identify interdependencies between the installation modules and the required CASE tools, software engineering techniques and training courses.

5.5.2 Training

There must be adequate training courses available to teach people the overall SDLC as well as the various tools and techniques used within it. The training should have specific courses aimed at the technical staff, systems management and the users. The training should also be pragmatic, using real world examples as much as possible.

(Note: Training should be conducted by people who have actual experience in using what they are teaching - not just "talking heads" familiar with the course notes.)

5.5.3 Guidance

You do not learn how to do something in a class room. You really learn it by doing it, preferably under the guidance of experienced people. The SDLC should provide guidance on how the guides will be trained and on how they should perform their duties.

Chapter 6 SDLC Design

Given the SDLC's objectives and requirements, there are certain architectural level design approaches which seem better suited to meeting the stated needs. In some cases it would be impossible to meet the requirements via any other design. These key design points are:

- The SDLC must be an example of itself.

- The SDLC must use a layered/iterative approach.

- The SDLC must keep distinct what must be done from how it is done.

- The SDLC must be organized in a hierarchical manner.

- There should be automated support for the SDLC itself.

6.1 An Example Of Itself

The SDLC is a system used to analyze, design and install systems. Therefore, the approach it espouses should be applicable to itself. This should manifest itself in two ways: format and contents.

6.1.1 Format

Most SDLCs today recommend the use of a number of the software engineering techniques for displaying the processes and data of a system. Therefore, the SDLC itself should use these techniques. It could use hierarchy charts, data flow diagrams and structured English to describe the processes. It could use entity-relationship diagrams and data structure charts to describe the data produced.

There are many alternative structured techniques for displaying this information. The specific techniques chosen are less important than the fact that a set is used consistently throughout the SDLC.

6.1.2 Contents

If you use the SDLC it should result in a system with a certain structure. The grouping of functions and data should conform to the rules of the recommended structured design approach. The level of detail should also conform to those rules.

Tasks would be bound by such parameters as what they produce, who does them and when they must occur. For example, it would be desirable to have all of the testing tasks

in separate "modules" since these might be done by a separate test team. A similar case can be made for data analysis and data design, training, hardware installation and many other activities.

Attention must also be paid to the boundaries of automation. It is desirable to have the tasks delineated in enough detail so that specific tasks could point to the CASE tool that should be used to do it. Without this delineation it gets difficult to integrate the CASE system with the SDLC.

An area of special concern is the definition of data. It is rare that an SDLC treats the deliverables produced by the process as true data bases. The deliverables must be organized into subject data bases and defined down to the element level. The relationships - i.e., pointers - to the other data bases must be defined.

One constraint that most SDLCs fall into is the fact that the project's deliverables are usually paper documents. Paper data bases present some severe problems, primarily in the area of data redundancy. The SDLC should take the approach that it can be creating true, normalized data bases which reside in a project dictionary, also called the project encyclopedia.

If this data base approach is used, then the various documents just become reports off the data base. For example, a data conversion specification must contain the description of the old and new file formats. However, this information also exists in the design documents for the current system and the new system. This information could get out of sync. However, if it is in the data base just once and the document definition just does an "include" when it is generated, the problem goes away.

This data base view is also absolutely critical to fully realizing the potential of the CASE tools. The CASE tools integrate through this data base. Without it we are left with a set of stand-alone tools with little overall integrity.

6.2 A Layered Approach

There are only two basic architectures to an SDLC. One is the traditional sequential approach wherein you first do all the analysis, then all the design and then look at the installation issues. Analysis, design and installation are each in discrete phases. The other approach is the layered, sometimes called iterative, approach. In it analysis, design and installation preparation are each pushed down to a specific level of detail in a given phase. In the next phase they are all pushed down to a further level of detail.

The sequential approach just does not work in anything but the smallest project. It also does not reflect how people really do systems. A layered approach is required to minimize scrap and rework, improve planning, minimize risks and improve quality.

6.2.1 Minimize Scrap And Rework

Obviously, analysis drives design which in turn drives installation and production support. However, there is also a feedback loop here. The structure of the design may change the scope of analysis. For example, there are different considerations in defining the requirements for an on-line system versus a batch system. On-line systems need better ergonomics, are more sensitive to performance issues, and are more tightly linked and interspersed into the user's everyday work. Therefore, design decisions such as on-line versus batch, centralized versus distributed, automated versus manual, etc., greatly affect the process of analysis.

Similarly, installation issues can affect the design and the scope of analysis. For example, the requirements and design may be based on certain assumptions about the structure and contents of the desired data bases for the new system. The data conversion analysis and planning effort may reveal that those data bases cannot be created from the existing files without undue risks and costs. When the new design calls for integrated data bases where stand-alone files existed before, it is not unusual for the data bases conversion effort to cost as much as the rest of the entire project combined. Thus, the project might need to be scaled back.

Similarly, the needs for problem management may require that diagnostic probe points be built into the system. This can have a major impact on many aspects of the design.

In yet another example, when training is looked at, the team may decide that the best approach is to build a training mode into the production system. Again, this can have a major impact on the design.

We could easily provide examples of each aspect of design affecting analysis and each aspect of installation support and production support affecting analysis and design. There is a tight feedback loop among these activities. They must proceed in parallel and in sync.

6.2.2 Improve Planning

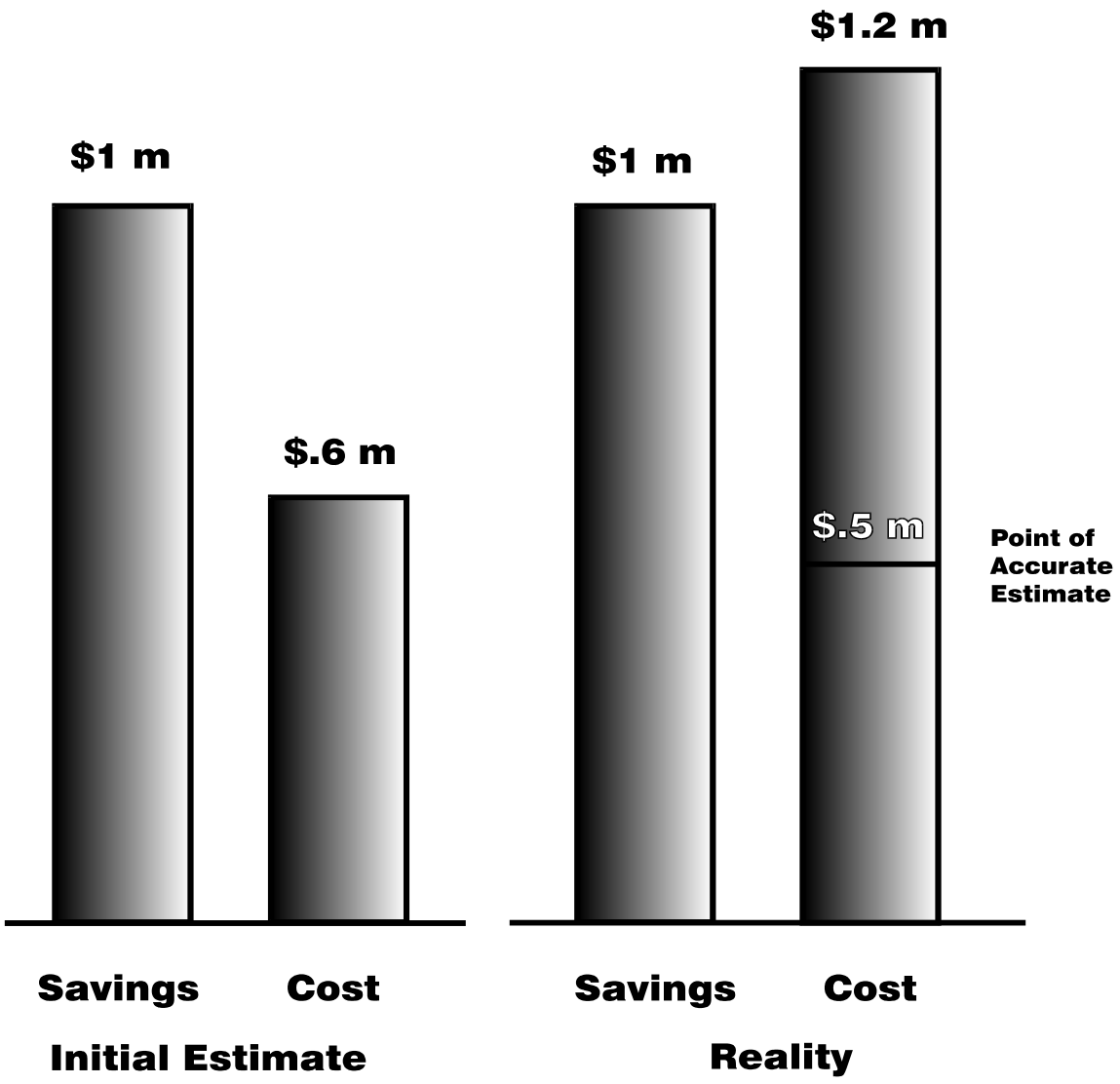
As stated in the planning section, the cornerstone of the plan is the completeness of the task list. Since projects are research and development, the tasks list cannot be completed until the research portion of the project is completed. As long as you are still doing research you do not know all of the tasks and, therefore, you do not know the costs or schedule.

If the SDLC is a sequential one, then you might still be doing research until near the end of the project. Many sequential lifecycles do not address the installation issues until coding is complete. These late discoveries may totally change the economic viability of the project.

If you look at Figure 2 you see the economic impact of the sequential approach. The initial projections showed a reasonable profit from doing the project. After a lot of money has been spent, new estimates revise this picture. However, the development cost to date is already spent money. It is gone. Do you now continue on or write off the project?

Figure 3 shows the basic difference between the sequential approach and the layered approach in regards to fully scoping the project. (In the figure equate "field support" with "production support".) Using a sequential approach you are 31/40ths into the project before you have completed the last bit of research. Using the layered approach you are only 4/40ths into the project.

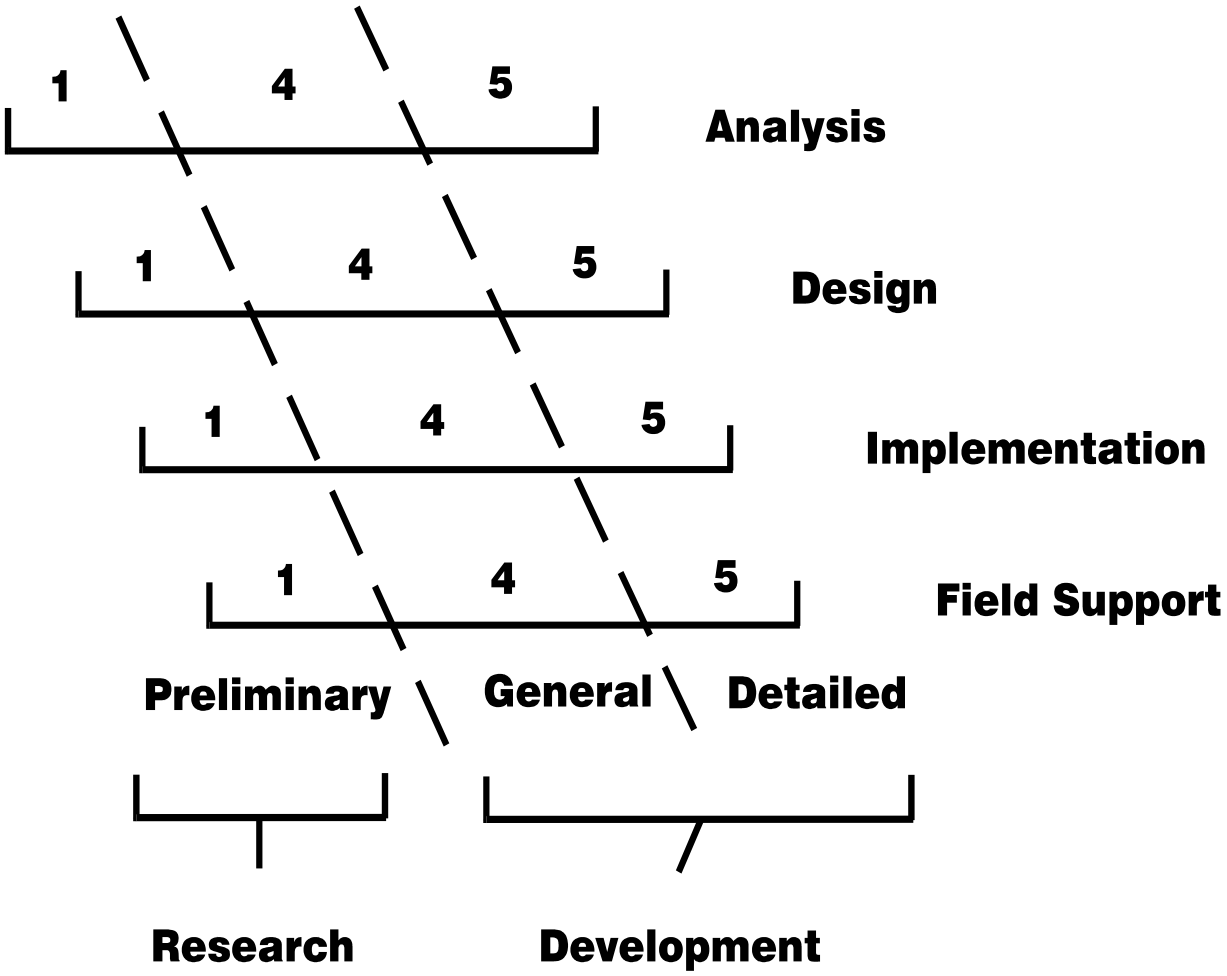
In actual experience, the research portion can be completed in about ten percent of the project's schedule but with just five to seven percent of the final resources. The project estimates are usually accurate to about plus or minus five percent. This is for staff experienced in using a rigorous layered approach.



Do you write off $500,000 or do you spend $700,000 more and lose only $200,000.

Figure 2

LAYERED APPROACH



Sequential Approach – 31/40
Layered Approach – 4/40

Figure 3

6.2.3 Minimize Risks

Unknowns represent risks. There are business risks, technical risks and people oriented risks as discussed earlier. By using a formal layered approach the unknowns are eliminated early and any difficult problems to overcome quickly identified. This gives the project team and the rest of the organization plenty of lead time to deal with them.

6.2.4 Improve Quality

The layered approach has a major impact on the quality of the product. By minimizing scrap and rework it avoids the temptation to make quick and dirty adjustments just to get something done. The integrity of the design structure can be maintained.

Perhaps the biggest impact it has on quality is allowing the project team the time to build quality into the product. Initial schedules are usually just someone's guess based on wishful thinking. However, the longer everyone assumes that that date can be made, the more it is locked in concrete. The rest of the organization begins making plans around that date. Later it becomes apparent to the project team that that date is unattainable with the full function at the desired level of quality. The team takes numerous short cuts to meet the date. The SDLC goes out the window, along with the rest of the software engineering techniques and CASE tools.

POOR SCHEDULES ARE THE SINGLE BIGGEST FACTOR IN POOR QUALITY.

6.2.5 Faux Layered Approaches

Many SDLC vendors who have sequential lifecycles just tell the client that they can execute the phases in parallel and get the same effect. This is blatantly not true. A layered SDLC must contain many specific tasks to keep the various products in sync during their parallel development. These tasks are missing from sequential lifecycles.

One area of the SDLC design where this is most glaring is in the design of the data. Sequential lifecycles tend to organize data into the "everything you know as a result of this phase" data base. In an application system this is equivalent to designing data bases solely on when the data was created. Thus you would have the 9AM data base and the 6PM data base.

In a layered approach the data structures must be clearly distinct from the phases. This allows the data bases to be updated as you go, both within a phase and across phases. (Kind of like a real system).

Another weak argument sequential lifecycles try to make is that there is a feasibility phase which scopes all these things out. Unfortunately, these feasibility phases are not very specific on how much detail is needed about each topic. They do not cover all of the research topics, usually omitting most or all the installation and production support

issues. They do not tell you how to do these high level tasks. They usually just state that you should follow the descriptions given for doing the detailed work. However, this is not always the best approach to the architecture level, nor is it easy to extract the high level tasks from the low level ones.

REMEMBER - EVERY FAILED PROJECT HAD A FEASIBILITY STUDY THAT SAID IT WAS VIABLE.

The layered approach supplies a truly rigorous way to determine the project's economic and technical viability.

6.3 Delineate What From How

The SDLC should clearly separate what tasks must be done from how they are done. The SDLC must clearly separate what information a task produces from how that information is displayed. This is required to ensure flexibility and maintainability in the lifecycle.

6.3.1 Flexibility

The best way to do a task or display information is dependent on the CASE tools available, the experience of the team in the use of the techniques, and the techniques being used on other, tightly related, projects. This distinction between what and how applies to all the technical tasks and the management tasks as well.

Some CASE tools themselves are context dependent. For certain categories of tools you need a different one for each programming language and/or DBMS and/or machine and/or operating system. Since these tools will often come from different vendors, they will reflect differing approaches to the same task. You may have to conform how a task is done to the tool you must use.

By decoupling the what from the how the project team has more flexibility in choosing the best way for them to get things done within the reality of their constraints.

6.3.2 Maintainability

The software engineering techniques are still evolving. The CASE tools are evolving rapidly. These tools and techniques describe how something gets done. However, with it all, the what has remained very stable. Plans still contain tasks, dependencies, resources, schedules and budgets. They have for literally centuries since the first formal projects were undertaken in construction efforts. What has changed is how we display the plan and the techniques used for creating it. The same can be said for analysis and design, etc.

If the SDLC clearly separates what from how, the impact of the (maintenance) changes is minimized. Everyone will still have a stable framework on which to hang the new tools and techniques.

6.4 Hierarchical Structure

Given the scope and complexity of the SDLC it will be a large system by any measure. This information is usually embodied in many manuals. The issue is organizing the materials so that the desired information can be quickly found at just the right level of detail. In the usability requirements section we discussed the various levels of expertise that the SDLC users will have. The expert user should not have to wade through piles of detail to get the overview checklists that they need. Novice users should be able to find detailed tutorials on topics that are new to them.

This requires that the information is layered in a hierarchy going from less detailed to more detailed as you go deeper. There would be a checklist level, possibly represented by a reference card. There could be several levels which define what is done. There would then be the lowest level which contains the tutorials on how to do something. This level could further point to the best reference books available on the topic.

6.5 SDLC Automated Support

SDLCs are often referred to as a beached whale. This is because the large number of binders, when opened to expose their rings, give the appearance of the ribs of some giant dead animal. It is also cumbersome to maneuver around with them.

What is desirable is that the information in the binders also be available on-line. The SDLC tool would allow an individual to quickly navigate through the appropriate levels of detail that they desire. There would be no need for everyone to have a large pile of manuals on their desk or to go hunt up the right manual when they needed it.

Having an automated version of the SDLC is also a major aid to the project manager. The SDLC is the superset generic task list, with dependencies and task overlaps identified. The SDLC should also have tailoring guides for fitting it to specific types of projects. If this information were available on-line the manager could quickly tailor the SDLC automatically via a project profile checklist. This could then be ported over to the project planning tool as the base line plan.

There already exist such tools in the market place (e.g., Nastec's Lifecycle Manager product) which allow you to load your own SDLC into it. However, the SDLC must be designed in a modular enough manner to make this possible.

Chapter 7 SDLC Project

It should be clear from the objectives that creating/selecting an SDLC is an important project. From the requirements and design points it should be clear that this is a large

and complex project. When you factor in the training and guidance issues it gets larger yet.

This is not a project which can be successfully executed via a part-time committee. It is a full time job. It also requires people who have the right level of expertise on this topic. Successfully producing systems using an SDLC does not guarantee that the person has the skill to create and install an SDLC. For example, you can successfully use a phone without being able to build one. You can successfully drive a car without being able to design one. The SDLC is complex enough that it takes years to learn how to build, install and maintain one.

The users for this project are the systems staff themselves. The user representatives to the SDLC project team must have a good understanding of their requirements and the impact of various design approaches. It is no different than their relationship with the users of the applications that they build and maintain. In the final analysis this is what this whole document has been about.

COMMON SDLC QUESTIONS

The following questions are those most often asked in introducing a new life cycle. They and their respective answers give some insight into the concerns and attitude of the staff.

1.We already have a life cycle. It did not work. Why will this time be any different?

> The existing life cycle was designed some time ago. As with any system, a life cycle can get out of date. The current one does not reflect the advances in software engineering. It also does not adequately support more complex systems (e.g. on-line, database. In addition, this time more attention is being paid to the installation support requirements -- training, guidance, tools. However, success will primarily depend on the level of commitment of everyone involved.)

2. This life cycle was not developed here specifically for us. How do we know it will meet our needs?

> This life cycle was selected after an analysis of our requirements. As with acquiring any package, we must do some level of tailoring to adapt it to our organization. The tailoring required should be relatively minor. The basic tasks and techniques required to build and support systems does not vary much from company to company, nor even from industry to industry.

3. The life cycle is too detailed. There is too much to read. I just want a checklist.

> Building systems is a complex process. Any book that tries to describe that process will be big. However, you do not need to read it all at one sitting. Read the overview material first. Then read the remainder as you need it. Once you are familiar with the material, the table of contents provides the checklist.

4. The life cycle is not detailed enough. It addresses what to do but not exactly how to do it.

> To address every "how to" would not be feasible. It would also be redundant with the many books that already exist covering the various techniques. The life cycle provides a context for the other books by showing how the pieces fit together into a coherent whole.

5. The terminology is confusing. It is not what I am used to.

Unfortunately there is no standard terminology in the industry today. A requirements document, functional specification, external specification and logical specification are all terms used for the document which describes what the user wants. The life cycle just had to select one set of terminology and go with it. As with learning the file names and data names in any system, the problem will go away shortly.

6. There are just too many tasks and too much documentation to produce. I do not have time to do all this. (Or) I do not need to do all this.

The life cycle provides the superset of the tasks and documents which must be considered. It must be intelligently subsetted and tailored to fit your project. For example, not every project will result in the acquisition and installation of new hardware.

7. (Variant of #6.) It would take too long on my type of project. Why should I use it?

The life cycle was designed with the various project types and sizes in mind. Again, it is primarily a case of proper subsetting. The alternative would be to have a different life cycle for every project type. This is unnecessary, costly and leads to numerous technical and business problems. If there is some aspect of the life cycle which makes it difficult to tailor it to your projects, let us know. We will work with you and fix it.

8. I believe this is all necessary, but my user will not give me enough time on the schedule to do it.

In building systems there are only three variables: time, function and resources. If we fix time, either function must be reduced or resources added. To produce the function in a reliable manner requires that we go through the tasks noted in the life cycle. Taking shortcuts increases the risk of not meeting the schedule and impacts the quality of the product. We need to work with the users to put the focus on their total objectives and not just dates.

9. I'd like to use all this but the time required to learn all this would impact our schedules too much.

In installing the life cycle and CASE tools, we must be sensitive to the effect of the learning curve. Therefore, you would rarely expect to install everything on a single project. We will start with those topics which will be most important to your project. We will also provide timely training and guidance to reduce the impact of the learning curve. In effect we are trading off the time you normally spend trying to figure out how to do the project -- basically defining your own life cycle. Therefore, the impact should be kept to a minimum.

10. How are you guys going to train us if you are still learning the life cycle?

Your focus will be on learning your users business needs and building a system to meet them. In the initial projects, our focus will be on learning the life cycle. Therefore, we should be able to stay ahead of your needs in this area. Also, we will get help from qualified outside sources as necessary.

11. My project has already started. Do I have to go back and redo everything?

No! We will use the life cycle as a checklist to ensure that there is nothing missing from what you have done already. Then we will use it to help you plan the rest of the project.

12. I could be doing more productive things. Why do I have to do all this?

First of all, you only do those things which are applicable to your project. (See answer to #6. Also, if you feel that there are more productive tasks that are not a part of the life cycle, let us know what they are. Either we need to fix the life cycle, or there is a misunderstanding of what it says.)

13. The life cycle puts a lot of emphasis on working with the user to get all their objectives and functional requirements. Won't that expand the project to the point where nothing will get done?

Again, this goes back to the trade-offs between time, function and resources. Also, if we are not getting this direction from the user it means we are making the decisions for them. In effect, we would be running their business, which is obviously inappropriate. Doing this up front also reduces the probability of significant changes in the scope of the project later.

14. My project is a low risk one. Why should I use the life cycle?

To make sure it stays low risk.

15. I am already doing most of this my own way. Why should I learn this life cycle?

If you are already doing most of this, it should make it easy for you to learn. The issue is that when each individual has their own way of doing things, it is hard to fit people into a coherent team. Also, on a given project you may (under the project pressures forget to do some of the things you know you should do. The life cycle acts as a repository of our collective experience.)

16. Why should I use it? Nobody else is.

In order to provide adequate support we will be installing the life cycle a few teams at a time. Eventually, everyone will be using it.

17. What's in it for me if I use it?

Learning the life cycle and the associated tools and techniques provides you with an opportunity to improve your professionalism. Also, the quality of the systems produced will be better. This means less scrap and rework to cover omissions, easier maintenance changes and fewer midnight calls to fix bugs. As the systems get more complex and require much higher reliability these skills will be necessary for your own continued success in this profession.

18. I have been successful for years without using a formal life cycle. Why do I need one now? (Alternative) This stuff is great for the future but why do I need to learn it now?

The skills and degree of formality required to build our current systems or install packages or perform maintenance are inadequate to build integrated, on-line, and/or database systems. The reliability, extendibility and usability factors are far more stringent. Also, given the amount of development to be done we must increase our productivity. If we don't start learning these techniques until we absolutely must have them, we will be overwhelmed by the learning curve. We will be learning new tools and techniques, new business functions and new technology. The risk would be too high. We must learn as much as possible while we are still working with familiar technology and functions (i.e. on our current projects. Plus, there are definite gains to be made in our productivity and systems quality even without the added impetus of the future needs.)

19. A formalized life cycle affects my creativity and individuality.

A formalized life cycle focuses the creativity into meeting the business needs. In that context the optimum in creativity is taking a very complex problem and providing a simple, cost effective solution. As for individuality, building systems requires teamwork. Your individuality is only affected to the extent required to work within the framework of a team.

20. I'm just a coder. Why do I need to know all this?

You don't. Your tasks are well delineated. More importantly, you can see how your tasks fit into the whole and who is producing the information you need to do your job.

21. My current system is already documented. Do I have to redo the documentation to fit the new formats?

No. The contents of the life cycle documents can be used as a checklist, though, to validate how complete your documentation is.

22. My current system is not documented. Should I go back and document it?

Use the life cycle documentation as a checklist to identify what you should have. Then separate what you must have from what would be nice to have. Generally, the key pieces of documentation would be a systems flowchart, a functional hierarchy chart cross referenced to the programs, and data layouts. The remaining documentation should be created as by products of making changes as opposed to making it a separate project.

23. Even as a project manager some of the tasks in the life cycle are outside of my responsibilities. How are they important to me?

The life cycle lists all the tasks which may be required to develop and install a system regardless of who will be assigned to do them. If a task is outside your responsibility, you should at least ensure that someone is addressing it. Otherwise, you may not be able to complete the project.

24. The life cycle calls for a lot of user involvement throughout the project. My user will not commit to that level of effort. What should I do?

> We are doing the system for the user. How do we know what they need without their help? If there is insufficient user support, you must evaluate the risks of proceeding. Generally, projects with complex business functions or which have a major impact on a lot of the user's personnel would be too risky without major support. The resources would be better spent on another project. Therefore, if you feel the risks are too great to proceed, you should escalate the issue to your management.

25. Does management really support this?

> Management realizes they cannot meet their business objectives without good systems support -- high quality systems built in a productive, cost efficient manner. The fact that they are investing time, money and resources in the life cycle, tools and training is evidence of the support for this effort.