
Trends in Systems Engineering Life Cycles

by Dr. F. G. Patterson Jr.

The essence of life cycle management is division of resources (6). Kingsley Davis recognizes division of labor as one of the fundamental characteristics of socialized man (9). The division of tasks into sub-tasks, the documentation of progress through intermediate products: these are important concepts that did not begin in abstraction, but in concrete problems. The essence of engineering is problem solving; and divide-and-conquer strategies, process definition and improvement, process modeling, and life cycle management are all engineering methods that begin by reducing complex challenges into tractable parts and conclude by integrating the results.

The failures of some systems engineering efforts have led some theoreticians to criticize or abandon the traditional ways of dividing resources. The trend is broad and can be seen in many areas. Life cycles are criticized for many reasons. For example, the concept that a requirements phase is self-contained, self-supportive, and separable from other phases has come under sharp attack both from academia and industry. The deeply ingrained tradition in industry that requirements must not dictate any given specific design is also coming into question. The justification for this belief may be readily derived from our basic activity model, shown in Figure 1.

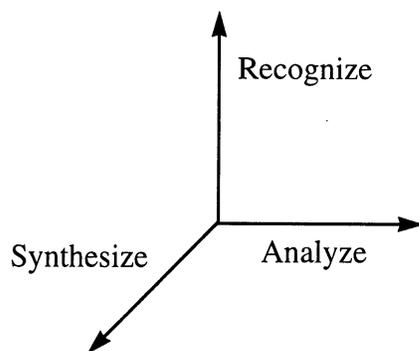


Figure 1. Engineering activity model.

Each of the three orthogonal activities shown in Figure 1 is necessary for successful systems engineering. It is not clear, however, that separation in time, separation by assignment to more than one action team, or separation in terms of any other resource based on the orthogonality of a process model (that is to say, a *life cycle*) is effective in reducing the engineering complexity problem. In fact, the opposite seems to be true. In the natural course of raising and resolving problems, it is much more efficient to resolve problems when and where they arise. The inefficiencies associated with following each life cycle step to completion before beginning the next step tend to result in a loss of information, and this tendency is exacerbated in proportion not only to the size of the engineering problem but also to the size of the engineering resources (5). This is a major obstacle in engineering big systems.

A solution based upon partitioning big systems into a number of small systems reduces the inefficiency inherent in life cycle-based approaches. However, this solution does not answer the emerging generation of systems engineering theorists who maintain that life cycle activities are dependent upon each other in a non-trivial way that neither a waterfall model, nor even a spiral model in its full generality, can adequately address. This suggests that specialties, such as *requirements engineering*, are undesirable, unless their scope of activity spans the entire life cycle: definition, development, and deployment. An alternate way to view this suggestion is that the number of specialties is effectively reduced to one: *systems engineering*. The systems engineer is ever-cognizant that a system must be specified, built, tested and deployed in terms not only of its internal characteristics, but also in terms of its *environment*. Thus, the job of integrating becomes a global concern that subsumes all of the steps in the life cycle.

Process Improvement

The trend away from the use of prescribed standard life cycles, while widespread in the private sector, has only recently begun to affect government acquisitions. The U.S. Department of Defense no longer requires the use of MIL-STD-499a for the acquisition of systems. Moreover, the new software development standard, MIL-STD-498 (10), after years of development, has been canceled without replacement. Other military standards are receiving similar treatment. The idea is that developers have their own processes that are specific to their own organizations. This change in emphasis should not be viewed as freedom from the use of a disciplined approach, but rather freedom to customize the approach to optimize quality attributes based upon variable factors in the software development organization.

The wide acceptance of ISO 9000 as an international standard is actually a trend away from standardized process models. The basic philosophy of ISO 9000 has been summarized as: "Say what you do; then do what you say" (14,23). ISO 9000 certification (15) is a goal to which many companies aspire in order to gain competitive advantage. Certification demonstrates to potential customers the capability of a vendor to control the processes that determine the acceptability of the product or service being marketed.

The Software Engineering Institute has developed a method of process assessment and improvement in the software development arena, known as the Capability Maturity Model (13,22,27). As the name suggests, the model is based upon the existence of a documented and dependable process that an organization can use with predictable results to develop software products. In effect, the details of the process are of little interest, as long as the process is repeatable.

The CMM model was adapted from the five-level model of Crosby (8) to software development by Humphrey. The five levels of the CMM model are:

1. The initial level: *ad hoc* methods may achieve success through heroic efforts; little quality

management, no discernible process; nothing is repeatable except, perhaps, the intensity of heroic efforts; results are unpredictable;

2. The repeatable level: successes may be repeated for similar applications. Thus, a repeatable process is discovered which is measurable against prior efforts;
3. The defined level: claims to have understood, measured and specified a repeatable process with predictable cost and schedule characteristics;
4. The managed maturity level: comprehensive process measurements enable interactive risk management;
5. The optimization level: continuous process improvement for lasting quality. According to Sage (25), "There is much double loop learning, and this further supports this highest level of process maturity. Risk management is highly proactive, and there is interactive and reactive controls and measurements."

The CMM helps software project management to select appropriate strategies for process improvement by examination and assessment of its level of maturity, according to a set of criteria; diagnosis of problems in the organization's process; and prescription of approaches to cure the problem by continuous improvement.

Even though managers may be seasoned veterans, fully knowledgeable about the problems and pitfalls in the engineering process, they may disagree with each other on how to cope with problems as they occur. If agreement is difficult to produce in an organization, the resultant lack of focus is taxing on organizational resources and may endanger the product. Thus the management of an organization must be greater than the sum of its managers by providing strategies for management to follow and tools for management to utilize. Such strategies and tools will be the result of previous organizational successes and incremental improvements over time, and measured by a level of maturity. The Capability Maturity

Model (CMM), developed at the Software Engineering Institute (SEI) at Carnegie Mellon University, provides a framework that is partitioned by such levels of maturity. Although the CMM was developed to measure the maturity of software development processes, the ideas upon which it is based are quite general, applying well to systems engineering and to such processes as software acquisition management, and even the software engineer's own personal software process (12).

In an analysis of the CMM, it is helpful to look closely at the five levels or stages in quality maturity attributable to Crosby (8) in order to see how one level builds upon another. They are:

1. **Uncertainty:** confusion, lack of commitment. "Management has no knowledge of quality at the strategic process level and, at best, views operational level quality control inspections of finished products as the only way to achieve quality."
2. **Awakening:** management wakes up and realizes that quality is missing. "Statistical quality control teams will conduct inspections whenever problems develop."
3. **Enlightenment:** management decides to utilize a formal quality improvement process. "The cost of quality is first identified at this stage of development which is the beginning of operational level quality assurance."
4. **Wisdom:** management has a systematized understanding of quality costs. "Quality related issues are generally handled satisfactorily in what is emerging as strategic and process oriented quality assurance and management."
5. **Certainty:** management knows why it has no problems with quality.

In each of these environments, a particular kind of person is required. There is a shift in focus from one type of key individual to another as we move from one CMM level to the next. The progression seems to be, roughly, as follows:

1. **Heroes.** Necessary for success in a relatively unstructured process, the hero is able to rise above the chaos and complete a product.
2. **Artists.** Building on the brilliance of the heroes, the artists begin to bring order, resulting through repetition in a codifiable process.
3. **Craftsmen.** These are the people who follow the process, learning from experience handed down from previous successes.
4. **Master craftsmen.** These are the people who are experts in their respective facets of the development process, who understand and appreciate nuances of process and their relationship to quality.
5. **Research scientists.** Finally, master craftsmen appear, who, through experiential learning and attention to process integration, are able to fine tune the process, improving the overall process by changing steps in the process while avoiding harmful side effects.

The characteristics of the organizational culture are directly related to organizational learning. The organization appears to depend primarily upon two factors: (1) the people who compose the organization, and (2) the environment internal to the organization. Of course, a case may be made for including the external environment, since the overall success of the organization (and its probability of survival) are directly related to its adaptation to both the market and technological factors. Some of the organizational characteristics may be organized in five stages, following the CMM model, as follows:

1. **Heroes and supporters.** Dependent upon the ability of heroes to rise above the chaos, the organization grows up around the activities of each hero, each of whom may require low-level support services. The hero's processes are largely self-contained, and very loosely coupled with other heroes' processes. While there are very efficient aspects of this kind of organization (viz., the hero's own activities), there is no overall efficiency induced by integration of activities

into an overall process. Thus, at this CMM level, there are really two levels of workers: heroes and others.

2. **Artist colony.** Through mutual respect and attention to the successful practices of the previous generation of heroes, these people work together to recreate the successes of the past, creating new processes along the way. Management begins to be able to track progress.
3. **Professional cooperative organization.** Through long service and attention to process, master craftsmen have emerged, creating more hierarchical structure in the organization as less experienced individuals are able to learn from more experienced craftsmen. There now exists the concept of “the way to do the job,” a concept that must be adhered to measurably. Management’s role is to control adherence to the process by defining metrics and implementing a metrics program.
4. **Society of professionals.** At this point, the organization is mature enough to be able to receive from its individual members meaningful suggestions on how to improve selected parts of its process and to implement them in the overall process. This is largely a shift in the organization’s ability to learn, of becoming a “learning organization.”
5. **Institute of professionals.** The organization is now so mature that it is able to look continuously for ways to improve processes. Outside influences are no longer repelled, but are welcomed and evaluated.

In parallel with the trend to decommission development standards, there is a trend to buy off-the-shelf products instead of customized or in-house developed systems. When products may be found in the marketplace that meet the requirements of customers, economy of scale in manufacturing may lead to substantial cost savings over the cost of custom development. Even when products are not available in the market, the trend among large customers, such as the U.S. Department of Defense and the National

Aeronautics and Space Administration, is to transfer the risk of in-house development from the customer to the contractor. Performance-based contracting is a tool that allows a customer to issue product specifications and acceptance criteria, and a supplier to collect a fee for creating the product as specified. Two significant differences between performance-based contracting and conventional contracting methods are:

1. There is very little or no oversight of the contractor by the customer.
2. All risks are borne by the contractor, who is paid upon delivery of a successful product.

Concurrent Engineering

Concurrent (or simultaneous) engineering is a technique that addresses the management of total life cycle time (7,24,26), focusing on the single most critical resource in a very competitive market: time to market (or time to deployment). This is accomplished primarily by shortening the life cycle through the realization of three engineering sub-goals:

1. Introduction of customer evaluation and engineering design feedback during product development.
2. A greatly increased rate of focused, detailed technical interchange among organizational elements.
3. Development of the product and creation of an appropriate production process in parallel rather than in sequence.

Concurrent engineering is a meta-process in which domain experts from all the departments concerned with developing a product at any stage of the life cycle work together as a Concurrent Engineering (CE) team, integrating all development activities into one organizational unit. The formation of the team does not, *per se*, shorten the engineering life cycle; however, through early involvement with the CE team, organizational learning and analysis activities

can be removed from the critical path to market. There is an explicit tradeoff of manpower for time to market. That is, the CE team involves more personnel for a greater fraction of the life cycle than in the case of the waterfall model. Although the CE team remains together for a greater percentage of the total life cycle, the life cycle is significantly shorter than in traditional models. Consuming a greater portion of a smaller resource may not increase cost and, in some cases, may actually decrease cost. However, the time to market may be greatly reduced. In terms of the abstract life cycle model, the activities labeled “recognize,” “analyze,” and “synthesize” can occur concurrently for all organizational elements involved in the development of the product. Of course, there will be some activities that have temporal, as well as logical, sequential dependence upon other activities (see Figure 2a and Figure 2b). Marketing, drawing upon organizational expertise, including RDT&E products, begins the process through the generation of an idea of a product, based upon market analysis. Marketing will generate targets for the selling price and the production costs of the proposed product to support management in deciding whether to proceed

with product development. During development, the CE team work simultaneously with the design team, to generate in parallel a design for the manufacturing process.

A CE life cycle model is shown in Figure 3. The principal feature of this process model is the concurrent development of the product, the manufacturing process, and the manufacturing system through the continuous participation of the CE team (28). A notable feature of the life cycle is the absence of a return path from production to design. This deliberate omission is in recognition of the extremely high risk of losing market share because of engineering delays due to design errors.

The organizational response to a change to concurrent engineering from traditional methods is likely to be fraught with difficulty. An organization that has formed around a particular life cycle model, and that has experienced a measure of success, perhaps over a period of many years, will almost certainly resist change. Effort in several specific areas appears to be basic to any transition:

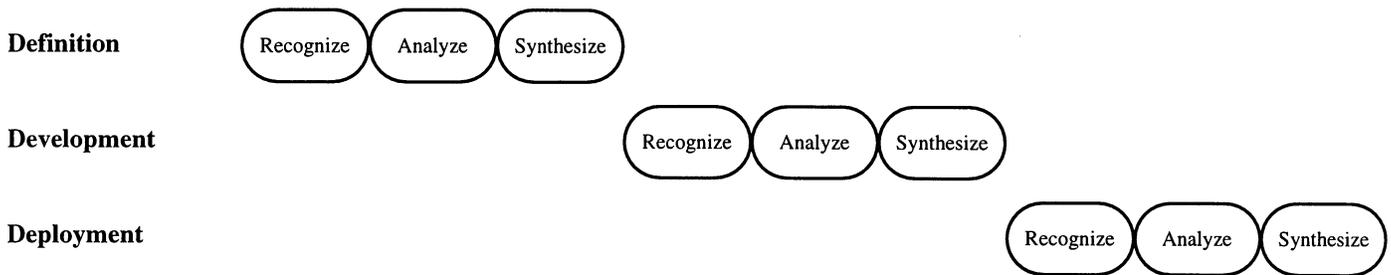


Figure 2a. Waterfall representation of abstract life cycle.

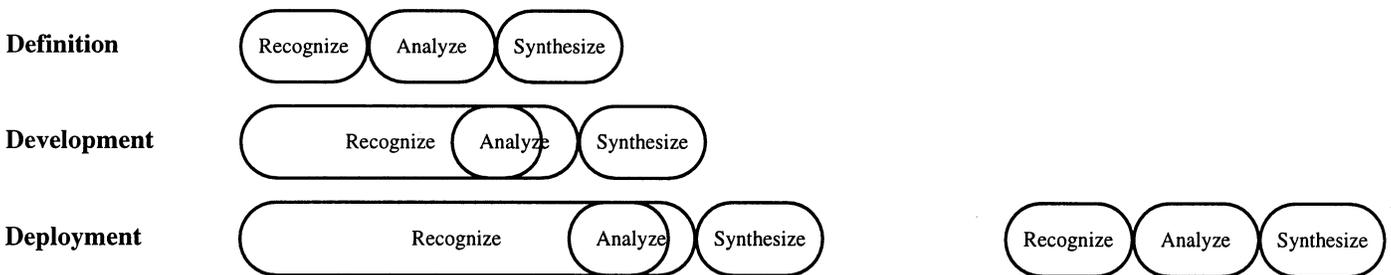


Figure 2b. The compressing effect of concurrent engineering upon the waterfall model.

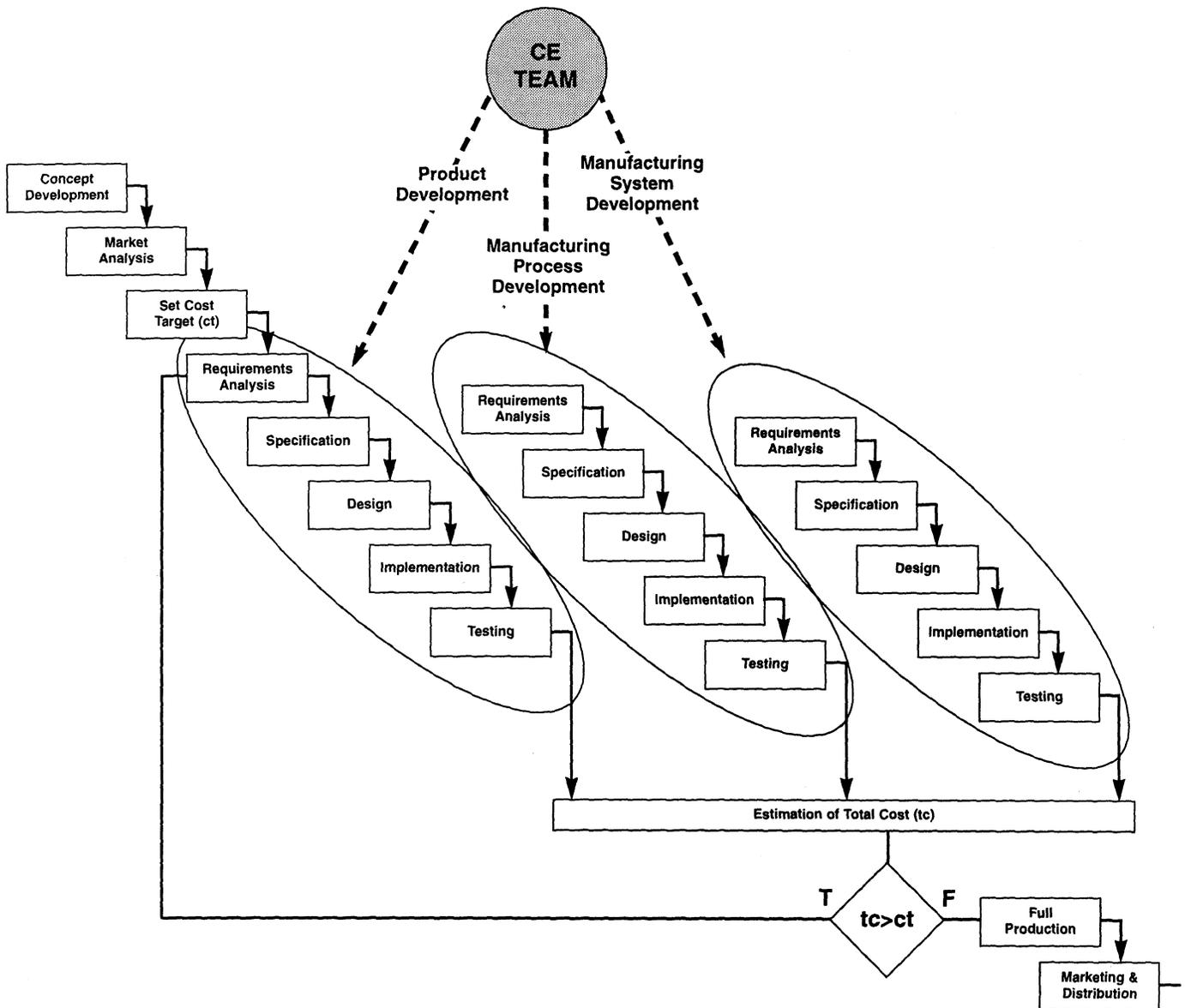


Figure 3. A concurrent engineering life cycle model.

- We have noted that life cycle models are purposeful, in that they reflect, organize and set into motion the organizational mission. A change of life cycle model should be accompanied by a clearly stated change of mission that may be digested, assimilated and rearticulated by all organizational elements — individuals, formal and informal team structures, and social groups.
- Formal team structures should be examined, destroyed and rebuilt, replaced or supplemented as necessary to conform to the new life

cycle model. Informal team structures and individual expectation, such as those described by Stogdill, may be replicated and thus preserved by the formal organizational structure to minimize loss.

- Particular attention should be paid to intramural communication and cooperation among individuals, teams, and departments in the organization (7). Communication and cooperation are essential elements of concurrency. Acquisition or improved availability of communications tools, developed through

advances in communications technology, may reduce the cost and increase the rate of concurrency.

- Software reengineering for reuse.

Based upon two types of reuse identified by Barnes and Bollinger (2), it is useful to distinguish between two different types of software reengineering for reuse:

1. Software reengineering for maintenance (adaptive reusability) improves attributes of existing software systems, sometimes referred to as legacy systems, that have been correlated to improvements that improve software maintainability (3).
2. Software reengineering for reuse (compositional reuse) salvages selected portions of legacy systems for rehabilitation to enable off-the-shelf reuse in assembling new applications (1).

Both types of reengineering for reuse share a common life cycle (20), shown in Figure 4, for reengineering to an object-oriented software architecture.

The life cycle is divided into three successive phases: reengineering concept development, reengineering product development, and deployment. During the concept development phase, reengineering is considered as an alternative to new software development. Considerations of scope and level of reengineering allow planning and cost estimation prior to the development phase.

Reengineering product development proceeds according to the scope and level of reengineering planned in the previous phase. Reverse engineering of the old software is followed by forward engineering to create a new product. During the reverse engineering stage, products are recreated at the design and specification levels as needed to recapture implementation decisions that may have been lost over the lifetime of the legacy software. During the entire reverse engineering stage, candidate objects are repeatedly created, modified or deleted as necessary to provide the basis of an object-oriented design for the forward engineering stage.

During the forward engineering stage, the candidate objects from the reverse engineering stage are used to create an object-oriented specification and design. Implementation through coding and unit testing complete the development phase. During the deployment phase, software integration and testing, followed by system integration and testing, allow production and deployment to proceed.

Software reengineering is often associated with business process reengineering. A recent study (21) shows that there is a reciprocal relationship between business process reengineering and software reengineering. The enabling role of information technology makes possible the expansion of the activities of business processes. Moreover, changes in support software may influence changes in the business process. In particular, changes in support software make the software more useful or less useful to a given business process, so that the business process

Reengineering Concept Development			Reengineering Product Development						Deployment		
			Reverse Engineering			Forward Engineering					
Feasibility	Scope	Level of Reengineering	From Code to Design	From Design to Specs	Identification of Candidate Objects	Respecification	Design	Coding and Unit Testing	Software Integration and Testing	System Integration and Testing	Production and Deployment

Figure 4. Software Reengineering Life Cycle.

(or, indeed, the software) must adapt. Changes in the potential for software functionality that are due to improvements in the technology may enable changes in business processes, but must not drive them. In general, successful technology follows, rather than leads, humans and organizations.

Similarly, changes in the business process create changes in the requirements for support software. However, this cause-and-effect relationship between business process reengineering and software reengineering cannot be generalized and is inherently unpredictable. Each case requires independent analysis. The effect of business process reengineering on software may range from common perfective maintenance to reconstructive software reengineering. New software may be required in the event that the domain has changed substantially. Because the software exists to automate business process functions, the purpose of the support software may be identified with the functions comprised by the process. Therefore, reengineering the process at the function level will in general always require reengineering the software at the purpose level. This is equivalent to changing the software requirements. Software reengineering can be the result of business process reengineering, or it can be the result of a need to improve the cost-to-benefit characteristics of the software. An important example of software reengineering that may have little impact on the business process is the case of reengineering function-oriented software products into object-oriented products, thereby choosing the more reactive paradigm to reduce excessive cost due to poor maintainability.

There are many levels of business process reengineering and of software reengineering, ranging from re-documentation to using business process reengineering as a form of maintenance. In both there is a continuum between routine maintenance (minor engineering) and radical, revolutionary reengineering. At both ends of the spectrum, change should be engineered in a proactive, not a reactive manner. As Sage notes, reengineering “. . . must be top-down directed if it is to achieve the significant and long-lasting effects that are possible. Thus, there should be a strong, purposeful and systems management orientation to reengineering” (25).

Lowry and Duran (18), in assessing the adequacy of the waterfall model, cite the lack of adequate support for incremental development, such as many artificial intelligence applications. The spiral model is much more natural, especially as computer-aided software engineering (CASE) tools shorten the production cycle for the development of prototypes. In terms of the spiral model (4), the amount of time needed to complete one turn of the spiral has been shortened for many of the turns through the use of CASE tools. A potentially greater savings can be realized by reducing the number of turns in the spiral as well as through the development of knowledge-based tools. Lowry believes that much of the process of developing software will be mechanized through the application of artificial intelligence technology (17). Ultimately, the specification-to-design-to-code process will be replaced by domain-specific specification aids that will generate code directly from specifications. This interesting vision is based upon much current reality, not only in the CASE arena, but also in the area of domain-based reuse repositories (11,16,19). In terms of the life cycle implications, the waterfall will become shorter, and the spiral will have fewer turns.

- (1) Robert S. Arnold and William B. Frakes, “Software Reuse and Re-engineering,” in *Software Reengineering*, R. S. Arnold, Editor. Las Alamitos, CA: IEEE Computer Society Press, 1992, pp. 476-484.
- (2) Bruce H. Barnes and Terry B. Bollinger, “Making Reuse Cost-Effective,” *IEEE Software*, pp. 13-24, January, 1991.
- (3) Victor R. Basili, “Viewing Maintenance as Reuse-Oriented Software Development,” *IEEE Software*, pp. 19-25, January, 1990.
- (4) Barry W. Boehm, *Software Engineering Economics. Prentice-Hall Advances in Computing Science and Technology Series*, R. T. Yeh, Editor. Englewood Cliffs, New Jersey: Prentice-Hall, 1981.
- (5) Frederick P. Brooks Jr., *The Mythical Man-Month: Essays on Software Engineering*. Reading, Massachusetts: Addison-Wesley Publishing

-
- Company, Inc., 1975 (reprinted with corrections, January 1982).
- (6) Vernon E. Buck, "A Model for Viewing an Organization as a System of Constraints," in *Approaches to Organizational Design*, 1971 paperback edition, J. D. Thompson, Editor. Pittsburgh: University of Pittsburgh Press, 1966, pp. 103-172.
- (7) Donald E. Carter and Barbara Stilwell Baker, *Concurrent Engineering: The Product Development Environment for the 1990s*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1992.
- (8) P. B. Crosby, *Quality is Free*. New York: McGraw-Hill Book Company, 1979.
- (9) Kingsley Davis, *Human Society*. New York: The Macmillan Company, 1949.
- (10) Department of Defense, *Software Development and Documentation (MIL-STD-498)*, 5 December 1994 edition. Washington, DC: United States of America, 1994.
- (11) John E. Gaffney Jr. and R. D. Cruickshank, "A General Economics Model of Software Reuse," in *14th Proceedings of the International Conference on Software Engineering*. New York: Association for Computing Machinery, 1992.
- (12) Watts S. Humphrey, *A Discipline for Software Engineering*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1995.
- (13) Watts S. Humphrey, *Managing the Software Process*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1989 (reprinted with corrections August, 1990).
- (14) Darrel Ince, *ISO 9001 and Software Quality Assurance*. Maidenhead, Berkshire, England: McGraw-Hill Book Company Europe, 1994.
- (15) International Benchmarking Clearinghouse, *Assessing Quality Maturity: Applying Baldrige, Deming, and ISO 9000 Criteria for Internal Assessment*. Houston, Texas: American Productivity and Quality Center, 1992 (American Productivity and Quality Center, 123 North Post Oak Lane, Houston, Texas 77024).
- (16) Wojtek Kozaczynski, "The 'Catch 22' of Reengineering," in *12th International Conference on Software Engineering*. Los Alamitos, CA: IEEE Computer Society Press, 1990, catalog number 90CH2815-2 (IEEE), p. 119 (26-30 March).
- (17) Michael R. Lowry, "Software Engineering in the Twenty-First Century," *AI Magazine*, volume 14, number 3, pp. 71-87, Fall, 1992 (Also appeared in *Automating Software Design*, eds. M. Lowry and R. McCartney, MIT/AAAI Press, 1991).
- (18) Michael R. Lowry and Raul Duran, "A Tutorial on Knowledge-Based Software Engineering," in *The Handbook of Artificial Intelligence*, volume IV, A. Barr, P. R. Cohen, and E. A. Feigenbaum, Editors. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1989, chapter XX.
- (19) John M. Moore and Sidney C. Bailin, "Domain Analysis: Framework for Reuse," in *Domain Analysis and Software System Modeling*, R. Prieto-Diaz and G. Arango, Editors. Los Alamitos, California, USA: IEEE Computer Society Press, 1991.
- (20) Floyd Guyton Patterson Jr., "Reengineerability: Metrics for Software Reengineering for Reuse," Ph.D. Dissertation, George Mason University (Fairfax, Virginia, USA), 1995.
- (21) Floyd Guyton Patterson Jr., "A Study of the Relationship Between Business Process Reengineering and Software Reengineering," *Information and Systems Engineering*, volume 1, number 1, pp. 3-22, March, 1995.
- (22) Daniel J. Paulish and Anita D. Carleton, "Case Studies of Software-Process-Improvement Measurement," *IEEE Computer*, volume 27, pp. 50-57, September, 1994.
- (23) John T. Rabbitt and Peter A. Bergh, *The ISO 9000 Book: A Global Competitor's Guide to Compliance and Certification*, 2nd edition. New