LIFE CYCLES FOR SYSTEM ACQUISITION

Patterson, F.G., Jr.  
George Mason University, Fairfax, VA., USA

Keywords: life cycle, acquisition, process, systems engineering.

Contents

1. Introduction  
2. Commonly Used Life Cycles  
3. Current and Future Trends  
Glossary  
Bibliography  
Biographical Sketch  
To cite this chapter

Summary

In this paper we look at a number of approaches for modeling the system acquisition process. Beginning with the waterfall model of Winston Royce, we continue with derivations that include more intricate and detailed waterfall models with feedback, including the “V” and spiral models. In addition we look at several acquisition strategies. In reviewing the various models we discuss our view that all of these process models are stages in the evolution of the system acquisition process. As technology improves, simplification of the process is becoming increasingly possible and affordable. Our evidence of this trend includes the increased use of concurrent engineering and concurrent design. In The Planning and Marketing Life Cycle we will examine models for this cycle and their relationships to the acquisition and the research, development, test, and evaluation cycles.

1. Introduction

This paper is concerned with life cycles designed to provide guidance for the management of resources applied to the engineering development of products. As such, life cycles provide the basis for a consistent acquisition or development strategy and are management tools. However, since the engineering of systems is routinely confronted with emergent properties that derive from the combination of their constituent parts, the study of life cycles is subject to many caveats that relate to the discovery, analysis, and treatment of these properties. Indeed, most variations and innovations in life cycle design seem to be addressed to this ubiquitous, challenging, and potentially fortunate problem.

1.1. Approaches

Three approaches are now widely recognized as strategies for product development under specific conditions: “grand design,” “incremental design,” and “evolutionary design.”
1.1.1. Grand Design

Grand design denotes a strategy for product design in which the product is designed in such a way that deployment is deferred until all development has been completed. Grand design is contrasted with incremental design and evolutionary design, both of which are described below. The grand design approach may be used for a product for which there is little expectation of requirements growth in the foreseeable future. Requirements growth may be dramatically reduced—or perhaps eliminated—by anticipating future needs. For example, in designing a new house, unused capacity can be intentionally built in, obviating the need for new design and construction at a later date. Requirements growth may also be limited by environmental, economic, political, or operational factors. An example would be rebuilding the Taj Mahal, which is constructed entirely of white marble—an unforgiving substance that is difficult to extend or change. Another example would be a single-use rocket, for which the lifetime of the operations phase would be a few minutes, quickly followed by the disposal phase. Yet another example would be a government-funded product for which future incremental or evolutionary funding might be extremely difficult to assure.

1.1.2. Incremental Design

Incremental development is a variation of the divide-and-conquer strategy in which a system is built in increments of functional capability. In this approach, as defined by Boehm, the first increment will be a basic, working system. Each successive increment will add functionality to yield a more capable working system. The several advantages of this approach include ease of testing, usefulness of each increment, and availability during development of user experiences with previous increments. Boehm's modification of the waterfall to allow incremental development provides a number of successive copies of each section of the waterfall model, starting with detailed design; followed by code, integration and product verification, implementation and system test; and ending with deployment and revalidation. The spiral model provides a more succinct representation in which each successive increment is assigned to the next higher level of the spiral. An example of incremental development is the International Space Station.

1.1.3. Evolutionary Design

The evolutionary development model is an attempt to achieve incremental development of products whose requirements are not known in advance. Boehm discusses a process that may be used with iterative rapid prototyping—especially automatic program generation—and user feedback to develop a full-scale prototype. This prototype may be refined and delivered as a production system or it may serve as a de facto specification for new development. More generally, where evolutionary development is possible, it can be represented using a waterfall model in the fashion used by Boehm to represent incremental development. However, representing evolutionary development entails repetition of the requirements design and specification development activities, and so substantially more of the waterfall must be repeated. Again, the spiral model is a more natural representation, since deployment for a given level of the spiral is directly linked.
to the definition portion of the next higher level, allowing necessary requirements growth in an orderly fashion.

Many examples of evolutionary design are found in software, such as accounting software, which must change with every new tax regulation. Another example of evolutionary design is the Icehotel in Jukkasjärvi, Sweden, which is entirely rebuilt out of ice each November and melts each May. In 1989 it was a one-room igloo, but it has evolved into a 60-room design and grows with the hotel’s requirements each year. It is interesting to note that the Icehotel could also serve as an example of grand design or incremental design, depending on several parameters. For example, since the structure melts each year, each “new” Icehotel is a grand design. However, since the design is reused from year to year, while perhaps adding rooms to the modular design, and since a part of the hotel is usable while the remainder is under construction, the hotel is also an example of incremental design. Finally, since many design decisions may be changed over time to add new functionality, such as the addition of a dining room or a chapel, the case for evolutionary design can be well made.

2. Commonly Used Life Cycles

The life cycle concept emerged as a model of the software process in the late 1960s, but it is now used more generally for the development of all types of systems. It is often argued that software engineering is a subdiscipline of systems engineering. Some authors call this “software systems engineering.” Software engineering, with its beginning in computer programming, has been slow to emerge as a discipline at all and, despite many improvements, is still regarded by many as more art than engineering. Nonetheless, software engineering standards have continued to evolve.

2.1. Waterfall Cycles

The waterfall model is the basis of all commonly-used process models. It is widely acknowledged that Royce presented the first published waterfall model (see Figure 1).

The elements of the acquisition life cycle may be cast into the definition–development–deployment structure (shown in Figure 2) that is the basis of many engineering models. We refer to this as the basic D–D–D model. The central idea is that each phase results in engineering products that spill over into the next phase. There are at least four basic reasons why the model is pleasing:

- The stages comprise the traditional systems engineering logic. Requirements analysis and specification fit into the definition phase. Design and implementation compose the development phase, while test and maintenance are comprised by the deployment phase.
- The stepwise nature of the waterfall suggests that only one phase is active at any one time, reducing the complexity of the engineering process.
- The logical ordering and the temporal ordering of the steps are identical.
- Perhaps most importantly, from an engineering management point of view, each step is represented as being largely self-contained, suggesting that different organizations may be designated for each phase without loss of continuity.
Since the introduction of the waterfall model by Royce, many other software and system process models have adopted a similar approach. The purpose of any model is to provide a framework that is better understood than the reality that it models and whose outcomes are sufficiently realistic to counter the objection: “Are things really this way?” Many variations of the waterfall model have been devised to overcome this objection.

Is Royce’s waterfall model realistic? Several problems in the use of this model have been observed. One problem is the duration of the phases: although phases tend to begin in the logically predictable sequence, in reality they never end. The products from each phase, if perfect, would indeed bring an end of the phase. A second problem is the suggestion that only one phase is active at one time. A third problem is the possible claim that separate organizations may have complete control of the project at different times in the development process without the loss of continuity.
Boehm presents a waterfall model (see Figure 3) that addresses many of these problems. It is based on nine products, the creation of which Boehm characterizes as sequential and which achieve necessary software engineering goals. The nine products are shown in Figure 4.

![Boehm’s waterfall model](image)

**Figure 3. Boehm’s waterfall model**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Concept</td>
<td>Concept definition, consideration of alternative concepts, and determination of feasibility</td>
</tr>
<tr>
<td>2. Requirements</td>
<td>Development and validation of a complete specification of functional, interface, and performance requirements</td>
</tr>
<tr>
<td>3. Architectural Design</td>
<td>Development and verification of a complete specification of the overall system architecture (hardware and software), including control and data structures</td>
</tr>
<tr>
<td>4. Detailed Design</td>
<td>Development and verification of a complete specification of the control and data structures, interfaces, algorithms for each program component</td>
</tr>
<tr>
<td>5. Code</td>
<td>Development and verification of a complete set of program components</td>
</tr>
<tr>
<td>6. Integration</td>
<td>Development of a working software system</td>
</tr>
<tr>
<td>7. Implementation</td>
<td>Development and deployment of a working hardware and software system</td>
</tr>
<tr>
<td>8. Maintenance</td>
<td>Development and deployment of a fully functional update of the hardware and software system</td>
</tr>
<tr>
<td>9. Phase Out</td>
<td>The retirement of the system and the functional transition to its replacement</td>
</tr>
</tbody>
</table>

**Figure 4. The nine products of Boehm’s waterfall model**

Boehm’s waterfall, similar to Royce’s, includes a return path from each step to the previous step to account for needed changes to the product of a previous step. Boehm’s
waterfall life cycle is logically complete, since it provides for the correction of deficiencies of previous steps. Moreover, it does so without violating the twin quality goals and constraints of configuration management and of validation and verification. The remaining inadequacies of this waterfall model are largely temporal and organizational.

As noted in our discussion of the Royce waterfall model, the logical and temporal characteristics of an engineering project may differ significantly from each other. In projects composed of several teams of engineers, the teams may progress at different rates. As a result, the temporal ordering of events may sometimes be reversed, except in cases in which the dependencies on which the logical ordering is based are of an essential nature. For example, an assembly of mechanical parts cannot be completed until each of the parts has been completed. The temporal and logical relationships between the assembly (the whole) and each part are identical and essential. However, the temporal and logical relationships among the teams’ completion of the parts may, in many cases, be independent of each other when the parts can be developed without design interaction with other parts. Moreover, a temporal problem may be created when a phase is deemed to have ended. For example, the requirements phase does, in fact, logically precede the design phase. However, the requirements work does not end when the design phase begins. It is of the nature of the waterfall model that each step begins with a set of requirements on which action is required. This set of requirements is the result of design decisions in the previous step, based on a previous set of requirements. Likewise, when the current step is complete, the result will be requirements for the following step. So, at each successive step in the waterfall, requirements beget requirements and changes to previous sets of requirements, the requirements for the whole governing the derivation of requirements for its parts. Every step is a requirements step and not one of them ever ends.

Figure 5. Four scenarios
To illustrate the organizational consequences of this problem over the lifetime of a project, at least four scenarios can be generated (as suggested in Figure 5), providing a
risk profile that depends on the size of the engineering problem or product and the size of the organization or team. In the figure darker shading depicts greater risk. For simplicity, we may characterize products, as well as teams, as large or small and examine each case in turn.

**Case 1.** This is the case in which a small engineering team has a small product to build. Provided that the team has both the engineering knowledge (skill) and the tools required, a small team can perform all life cycle tasks with great efficiency and with a very high probability of success. There are many success stories about significant systems that have been specified, designed, built, tested, and successfully marketed by entrepreneurs with limited resources. There are doubtless countless untold success stories about more typical efforts of small engineering teams.

**Case 2.** In this case, a large engineering team has a small product to build. Management may decide to exploit the small problem size to advantage by using the approach of Case 1. Alternately, several small teams may be engaged, thus introducing additional learning curve costs, necessitating the continuous employment and communication among a larger number of engineers, lowering the probability of successful completion and increasing risk.

**Case 3.** When a small team undertakes a large engineering task, management must make several decisions based on available resources. If time is not a critical factor, a small team approach offers the high quality, low risk, and economy of Case 1. In effect the large problem has become manageable by subdivision into smaller problems without introducing the complexities of using subcontracts. Unfortunately, time is almost always a critical factor and management is forced into the role of general contractor, assuming the high risk of using independent contractor teams, each with its own layer of management, its own development process, and its own strategic goals.

**Case 4.** If the large team has the required skills, methods, and tools, performing all tasks in-house may mitigate the high risk of Case 3. In this way, a single development process and a single set of strategic goals improve communication and cooperation among groups of engineers. However, large organizations are inherently plagued by problems of management and communication complexity. Moreover, because of the need to retain the organizational expertise from previous steps at each successive step in the life cycle, the total organization may be very large—and largely unoccupied—at any given time.

Sage describes a 22-phase acquisition life cycle model (see Figure 6), organized into three general divisions:

- definition,
- design and development, and
- operations and maintenance.

The model is very comprehensive, including activities from the first perception of need for the product through to maintenance activities after system deployment.
SYSTEM DEFINITION
1. Perception of Need
2. Requirements Definition
3. Draft Request for Proposal (RFP)
4. Comments on the RFP
5. Final RFP and Statement of Work
6. Proposal Development
7. Source Selection

SYSTEM DESIGN AND DEVELOPMENT
8. Development of Refined Conceptual Architectures
9. Partitioning of the System into Subsystems
10. Subsystem Level Specifications and Test Requirements Development
11. Development of Components
12. Integration of Subsystems
13. Integration of the Overall System
14. Development of User Training and Aiding Supports

SYSTEM OPERATION AND MAINTENANCE
15. Operational Implementation or Fielding of the System
16. Final Acceptance Testing
17. Operational Test and Evaluation
18. Final System Acceptance
19. Identification of System Change Requirements
20. Bid on System Changes or Prenegotiated Maintenance Support
21. System Maintenance Change Development

Figure 6. Sage's 22-phase acquisition life cycle model

The first set of activities in this life cycle begins with the first formal response to the recognition of need and ends with a system specification and a systems engineering management plan. Collectively these activities are referred to as the definition phase of the acquisition cycle. The system specification—the first formal description of the product—guides the development phase.

In the second division of the acquisition cycle, referred to as the development phase, a system architecture is created in concert with the system specification and the system engineering management plan. Both the specification and the plan determine the architecture of the product and the structure of the engineering organization. Thus, the derivation of architectural detail is as a management as well as an engineering function. Related engineering activities include the identification of subsystems and the beginning of configuration control. The next lower level of detail comprises the specification and design of subsystem components, including hardware, software, firmware, test equipment for hardware testing, and scaffolding (a simulated operational environment used for software unit testing). Fabrication and testing of hardware components, as well as the coding and unit testing of software and firmware components, begins after approval of the design. Integration of components into subsystems is the next step, which involves not only integration testing, but also validation of the built and tested subsystems with respect to subsystem specifications. Next, the subsystems are integrated into a system that may be integration tested and validated against the system specification. Integration test procedures are a product of this phase, which may be useful in the system maintenance part of the system life cycle. When system testing is complete, attention must be given to other aspects of preparing to deliver the system,
such as installation materials and procedures, training manuals and special training equipment and procedures, maintenance equipment and manuals, and user’s guides.

**Figure 7. The 32-step systems acquisition life cycle**
Deployment is the third part of the acquisition life cycle, beginning with the physical delivery and installation of the system and ending with the decision to decommission or replace the system. As in the case of the other two phases, the deployment phase may be viewed as a process with its own life cycle. The first phase of the deployment life cycle begins with delivery and installation, acceptance testing, operational testing and evaluation, and acceptance of the system by the customer. The second phase accommodates changes to the system, such as software maintenance. There may be a formal maintenance agreement with a contractor that includes a maintenance plan, maintenance test plans, and maintenance test procedures. The third and final phase of the deployment life cycle is the decision to remove the system from operation. This phase is often initiated by a tradeoff study to compare options for continuing to support the business process. Our description of the systems engineering life cycle is summarized in Figure 7.

The 32-step systems acquisition life cycle shown in Figure 7 can be constructed using the D–D–D framework of Sage. In turn, applying the D–D–D structure recursively to each of the three Ds provides further details. The first D is the definition phase, which begins as in Sage’s 22-step cycle (Figure 6) with the perception of need. There are three important outputs: system requirements, specifications, and the systems engineering management plan, as well as various contractual vehicles and other transition documentation.

A requirements statement for a product or service is an informal expression of need, elicited from a person or group of persons who will be affected, either positively or negatively, by the product or service. The requirements are the most accurate and robust representation of the stakeholder needs. Quality in terms of customer satisfaction should be measured by using the requirements, and not the specification, as the standard definition by which the resulting products or services are validated. However, requirements characteristically suffer from problems that originate in the imprecision usually attributed to natural language. Specifications are formal engineering products that mark the end, rather than the beginning, of the requirements life cycle. A list of seven steps in the requirements phase is given in Figure 8 and the requirements life cycle is illustrated in Figure 9.

**DEFINITION**
1. Perception of Need
2. Elicitation of Requirements

**DEVELOPMENT**
3. Classification (organization)
4. Requirements Analysis
5. Prototyping

**DEPLOYMENT**
6. Specification of Requirements
7. Preparation of Transition Documents

Figure 8. The seven steps in the requirement phase
There are many approaches to developing a list of requirements. The list may be generated using a team approach or, for smaller projects, by interviewing stakeholders individually to discern individual and collective needs. In either case, the starting point of the requirements engineering process is normally an elicitation process that involves a number of people to ensure consideration of a broad scope of potential ideas and candidate problems.

The next three steps (also shown in Figure 8 as steps 3–5) provide understanding of the requirements and improve the quality of the design decisions that the requirements represent. As a result, each step depends on substantial communication with the stakeholder community. Step 2 is the organization of the requirements. In this step there is no substantial, interpretive transformation of the requirements, but only simple classification and categorization. For example, requirements may be arranged into functional versus non-functional groups or into objects and object classes. Much more substantial change occurs in the third step: analysis of the requirements. For example, details may be added that differ significantly from the perceptions of the stakeholders. During the analysis step, the problem areas addressed might include ambiguity, completeness, conflict or consistency, correctness, performance, redundancy, reusability, and testability. The fourth step in the requirements development life cycle is prototyping. The development of a prototype is a synthesis step in which poorly understood parts of the problem are developed to some level of completion, usually to answer specific questions about the design or to elicit specific design decisions from the stakeholder community.

In the fifth phase of the requirements process, the stakeholders’ requirements—substantially improved through classification, analysis, and prototyping—are transformed into a specification. The specification is an engineering product. It is the first complete, although abstract or “high level,” representation of the product under development. The specification will undergo continual modification as design decisions are made throughout subsequent phases of the acquisition life cycle. Details will certainly be added, and important changes may be made to the requirements and reflected in the specification as emergent properties of the design are discovered during the development process.
The transformation from a raw list of stakeholders’ requirements to a specification introduces a significant risk in the engineering process. As the requirements that were elicited from the user and refined through analysis, synthesis, prototyping, and feedback are massively reformed into a specification, there is a transition of substance (through adding of details necessary to support successful implementation) and of language (through the use of carefully stated natural language and other representations, such as state charts and various mathematical notations).

The system specification is the engineering definition of the product to be developed or acquired. In the development phase the specification is partitioned, enlarged, and refined in accordance with the detailed life cycle that is used. Typically, the specification is decomposed in the way that best utilizes the resources of the development organization, unless specific guidance for decomposition is provided in the requirements definition. For systems that are primarily composed of hardware, the benefits in terms of cost and quality of allowing the development organization a high degree of freedom in interpreting the specification are passed along to the customer. Software systems differ in that the product architecture must support the evolutionary nature of software maintenance—the part of the software life cycle that predominates in terms of both duration and cost. One of the principal advantages of in-house software development, as opposed to acquisition of either custom or commercial-off-the-shelf (COTS) software is that, because the development organization and the maintenance organization are the same, product cost and quality dividends are effectively realized twice: during development and again during the maintenance phase, over a much longer time period.

There are several additional interesting points to be noticed about the requirements life cycle as depicted in Figure 9. First, we refer to the definition phase of the systems engineering life cycle as containing the requirements life cycle. That the requirements life cycle is a “proper life cycle” may be seen by noting that each of the steps is well defined, each with a product that characterizes the step. Moreover, the steps may be grouped into the familiar D–D–D paradigm. Requirements elicitation provides the raw requirements. Classification, analysis, and prototyping develop the requirements, which are then deployed as a requirements list and transformed into a requirements specification. None of this is to say, however, that activities that have been ascribed to one phase or step, rather than to another, may not be performed or revisited in a subsequent phase or step. Decision making is the essential nature of engineering; the more we know about a problem to support decisions, the more we may define, develop, and deploy. Thus the partitioning of activities into phases, and the resultant linear ordering of activities, are done to facilitate understanding, but certainly not to levy boundary conditions on phased process activities. Second, we may note that the more transformations that are performed on the elicited requirements, the less communication is possible with the stakeholders. This constraining influence represents risk, in that the stakeholders may be unaware until a failure occurs of possible misapplication of requirements. Figure 9 depicts lines of communication with the stakeholders using solid and broken lines, where solid lines represent relatively good communication, and increasingly broken lines represent increasingly poor communication. Third, we may note an association among the first three steps. Figure 9 frames these steps together and shows solid lines of communications to the stakeholders. This suggests that these
activities may all be done together as a stakeholder team activity, thus further investing the stakeholders in the process of developing high quality requirements. Because of the great potential benefits associated with keeping the stakeholder team involved, it is important to discipline the requirements engineer to use the language (terminology, examples, associations among requirements, etc.) of the stakeholder and to refrain from introducing engineering models, templates, paradigms, and tools during stakeholder involvement.

In addition to the requirements cycle elements, the definition phase of the acquisition life cycle contains linkages to subsequent activities in the development phase and to other enterprise activities. The other enterprise activities are elements of the planning and marketing cycle and the research, development, test, and evaluation cycle (RDT&E).

Acquisition documents create a link within the acquisition life cycle between the requirements definition process and the development process. As shown in Figure 10, the set of documents includes both technical specification and contractually negotiated documents. In the case of in-house development, the requirements specification may be all that is required. In most projects, except for those carried out within a single organization, the document set includes a request for proposal and a proposal, containing a statement of work, a systems engineering management plan, and a work breakdown structure.

Figure 10. Acquisition documents

The end of the deployment life cycle as depicted in Figure 7 is very much linked to the planning and marketing process. This part of the life cycle is concerned with how a deployed system is used to support a business process in some organizations. In step six the concept of a tradeoff study to compare options for continuing to support the business process is a very general notion. In the case of a manufacturing process, market factors
are the primary criteria for continuing, renovating, or retiring and replacing the system that supports the process. More subtly, in the case of nonmanufacturing processes, the performance of the organization is the primary consideration. In both cases, change in the business process drives change in the system that supports it, although technological improvements alone may result in changes, especially where improvements in economy, reliability, or safety are possible.

Linkage with the planning and marketing cycle may include requirements that influence product definition, especially those that incorporate market research products. Input from the planning and marketing cycle is critical to setting and maintaining cost goals. Output from the acquisition process to the planning and marketing process allows early product information to be used to create marketing plans, tools, and schedules.

Linkage with the RDT&E process may be of two basic types. If the research was directed by the requirements development process either to clarify requirements or to solve a technical problem, then the research results are needed to complete the effort. On the other hand, if the research product was the result of independent and innovative work of an on-going nature, through which the organization creates new products, then the product definition must be handed off to the requirements team to allow the product to be brought to market.

Linkage from the development phase to the deployment phase (see Figure 11) consists of documentation to support the organizational, logistical, and technical aspects of bringing the engineered system into operation; the full development suite of detailed technical documentation, including specifications, designs, drawings, and other data that will support system maintenance; and operational information, such as user’s guides, and training to support operation of the system.

Figure 11. Support documentation
2.2. Variations of the Waterfall Model

Most systems are designed from the top down and then implemented from the bottom up. Ould graphically represents this approach as the “V” process model for software engineering. Figure 12 depicts the “V” as a composition of three layers or views of the system in increasing engineering detail. The top view is the customer view, which associates the total system requirements with the delivered system that the requirements document represents. The customer view is the perspective of the user or stakeholder, who is interested in submitting a list of requirements and receiving a finished product that meets the requirements.

A principal strength of the “V” depiction of the waterfall approach is the ability to represent validation and verification as horizontal and vertical relationships. In the customer view, the requirements documentation is validated horizontally against the fully integrated system. In contrast, the inspection and analysis of the decomposition of the system requirements into segments or subsystems in the second layer act as verification, which is depicted vertically. According to ANSI/IEEE STD-729, software verification is “the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase.” The horizontal dimension asks a different question, whether the built and tested product at some level of completion successfully represents the specifications and designs at the same level of the model: this is validation. Again according to the
ANSI/IEEE standard definition, software validation is “the process of evaluating software at the end of its software development process to ensure compliance with software requirements.”

The second or middle layer of detail is the architectural model or the classical systems engineer view, which, on the descending (left hand) side of the “V,” addresses the decomposition of the system level specification into system design and subsystem specifications and designs, which are broken down further into component specifications and designs. On the ascending (right hand) side of the “V,” components are grouped together to compose subsystems, which in turn are integrated to create the finished product. The architectural model is the traditional perspective of the systems engineer, who is interested in decomposing the whole into manageable parts, respesifying and designing the parts, and integrating the parts to compose the complete system. Both definitions, written specifically for software engineering, extend in full generality to apply to the engineering of all systems.

The third and lowest level is the implementation model or builder view, which couples component specifications and designs with fully tested components. The implementation model is the perspective of the component developer or contractor, who is interested in optimized component level specifications, designs, and products. In the “V” model, component level specifications on the descending leg are associated horizontally with and validated against the manufactured components on the ascending leg.

Forsberg and Mooz have developed a much more comprehensive “V” chart, including a replicated “V” for incremental and evolutionary strategies. Ould also describes a variation of the “V” process model known as the “VP” life cycle, where “P” refers to prototyping. When requirements are not well understood for system elements, prototypes may be developed to provide insights and answers to problems and questions. The use of a prototype at any point in the life cycle may be considered to be a form of directed research and development.

When Boehm devised the spiral model (see Figure 13), he addressed not only the division of the problem, but also the division of the engineering resources needed to solve the problem. We have looked at the structure and the function of the waterfall. An insight into the nature of the possible problems with the waterfall can be gained by reflecting on the purpose. Life cycle models represent an attempt to mitigate complexity by applying a divide-and-conquer strategy to the problem of managing engineering resources. The ordering of processes and products into phases—the organizational structuring—is necessary to managing the complexity of the solution. However, dealing exclusively with the resources assembled to solve the problem cannot reduce problem complexity. The problem itself must also be “re-architected” into a manageable form and size, where manageability varies according to whether or not the domain is well understood. The waterfall model can be applied well to manageable problems in familiar domains.
As previously noted, repetition is a key feature of Boehm’s spiral model. The spiral has been compared to a repeated waterfall model and to a repeated “V” model. As shown in Figure 13, the spiral model repeats a common pattern during each turn of the spiral. As suggested in Figure 13, Boehm’s abstracted pattern is comparable to the D–D–D pattern identified by Sage. Boehm divides the model into four quadrants (see Figure 13). Quadrant I is the beginning of each level of the spiral and is concerned with identifying needs, possible ways of satisfying the needs, and factors that may constrain the solution. Quadrant II represents the follow-on to the requirements identification of Quadrant I, during which requirements are analyzed, problems are resolved, and tradeoffs are made. These two quadrants represent the parts of the waterfall model concerned with definition of the system to be built. In Quadrant III, a partial product is developed and verified. Quadrant IV completes the turn of the spiral by deploying the partial product either as a finished product or as a partial product together with a plan for its completion during subsequent turns.

The spiral model implements very naturally the incremental and evolutionary development approaches. The waterfall process model can be seen to be most appropriate for the development of products in a familiar domain, where unexpected emergent properties of the design are unlikely to appear. For unfamiliar domains, and in large or complex projects, an incremental approach reduces risk, since the cost of each increment is relatively small. An increment may be redeveloped or perhaps even discarded without side effects that have catastrophic cost consequences. The great strength of the spiral model is the capability to develop increments—or prototypes—with each full turn of the spiral. The prototype that is specified, planned, built, tested, and evaluated is now a stable core element of the final system.

2.3. Concurrent Engineering Cycles

Perhaps the most important resource in a highly competitive product design environment is time to market. Concurrent engineering is a technique that addresses the
management of total life cycle time, primarily by reducing the length of the life cycle through the development of the product and creation of an appropriate production process and facility in parallel rather than in sequence, a greatly increased rate of focused, detailed technical interchange among all organizational elements, and increased customer evaluation and engineering design feedback during product development.

Concurrent engineering is an approach to shortening the life cycle in which experts—from all the disciplines needed for 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. Through early involvement with the CE team, some 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 the same number or more personnel for a greater fraction of the life cycle than in the case of the waterfall model, while reducing the total time to market for the overall product development. There will generally be some activities that have temporal, as well as logical, sequential dependence on other activities, allowing little or no reduction in total time for those efforts. However, where many non-sequential reprogramming opportunities are found, savings in total time may be very significant.

A CE life cycle model is shown in Figure 14.

![Figure 14. A concurrent engineering (CE) life cycle model](image-url)
Marketing, drawing on organizational expertise, including RDT&E products, begins the process through the generation of an idea of a product, based on 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 works simultaneously with the design team, to generate in parallel a design for the manufacturing process. The principal feature of this life cycle model is the concurrent development of the product, the manufacturing process, and the manufacturing system through the continuous participation of the CE team. 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.

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.

3. Current and Future Trends

During the 1970s and 1980s, the acquisition life cycle was widely believed to be an effective temporal model—a recipe—rather than a logical model of engineering activities. This means that each stage or step in the life cycle was thought to be largely self-contained, accepting inputs from the previous stages and creating outputs for the following stages. When it was recognized that early design decisions during the requirements phase constrain the designer during the design phase, thereby removing the freedom to consider alternatives and often preempting designs that are superior, a management trend developed that would prove to be very harmful. Management attempted to maintain strict separation between the life cycle phases by prohibiting any design work prior to or during the development of system requirements. In this way it was thought that designers would not be constrained in the design phase in counterproductive ways. In support of the trend, much was written about the differences between requirements and design. Requirements engineers were often evaluated by management’s perception of their ability to write such requirements that did not constrain design.

Progressively in the 1990s it became widely recognized that many of the system design efforts of the previous decades had failed despite very strict adherence to the acquisition cycle qua temporal model. In an overreaction to the failures, many organizations discontinued the use of life cycle models and in some cases even prohibited their use. The rationale usually followed the theme of allowing the developing organization the freedom to use its own, presumably successful, processes rather than process models imposed by the customer. Thus, as we shall see, the acquisition contracting problem was transformed from that of choosing the best life cycle for the development of the product to that of measuring or certifying the maturity level of the contractor’s own process model. Environmental management goals, such as ISO 14000 conformance (see The Planning and Marketing Life Cycle), should be designed into acquisition processes under any of the following standards.
3.1. ISO 9000

ISO 9000 was first published in 1987, then revised in 1994, and finally updated in 2000. The wide acceptance of ISO 9000 as an international standard is 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.” ISO 9000 certification 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.

3.2. The Capability Maturity Models

The Software Engineering Institute (SEI) has developed an integrated family of models—known as capability maturity model integration (CMM or CMMI)—on which methods and tools for process assessment and improvement in the software development arena have been based. As the name suggests, the models are based on the existence of a documented and dependable process that an organization can use with predictable results to develop products in a specified discipline. The first of these was the CMM for software engineering, which has had a broadly positive effect on software engineering organizations that have implemented the model. In effect, the details of the organization’s software engineering process are of little interest, as long as the process is repeatable.

The CMM model was adapted from the five-level model of Crosby to software development by Humphrey. The five levels of the Crosby and CMM models are shown together with the staged and continuous integrated CMM representations in Figure 15. The CMM helps 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 continual 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 CMM developed by 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 on which it is based are quite general, applying well to systems engineering and extensible—applying to such processes as software acquisition management and even to the software engineer’s own personal software process.
Figure 15. The Crosby, CMM, and staged and continuous CMMI models

With CMMI-SE/SW V1.0, organizations currently using different models for separately improving systems engineering and software engineering will be able to use the integrated model to improve, train, and assess processes more commonly and consistently. Use of this CMMI model supports enterprise-wide improvement by enabling the integrated assessment of both systems and software engineering. The integration model incorporates the best features of the source models from which it is derived. This new model will enable organizations to build on previous investments in improvement that were based on the SW-CMM V1.1 or the SECM, and at the same time to benefit from the standardization and commonality of the integrated model.

In analyzing the CMM, it is helpful to look closely at the five levels or stages in quality maturity attributable to Crosby. They are:
• 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.”
• Awakening. Management wakes up and realizes that quality is missing. “Statistical quality control teams will conduct inspections whenever problems develop.”
• 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.”
• 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.”
• 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 shifting 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:

• Champion. Necessary for success in a relatively unstructured process, the champion is able to rise above the chaos and complete a product.
• Artists. Building on the brilliance of the champions, the artists begin to bring order, resulting through repetition in a codifiable process.
• Craftspeople. These are the people who follow the process, learning from experience handed down from previous successes.
• Master craftspeople. These people are experts in their respective facets of the development process, who understand and appreciate nuances of process and their relationship to quality.
• Research scientists. Finally, master craftspeople 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.

3.3. SPICE

Launched in 1993 and scheduled to be published as a full international standard in due course, SPICE (ISO/IEC 15504) is an international effort to develop a standard for software process assessment, organized by the International Organization for Standardization (ISO) International Committee on Software Engineering Standards (ISO/IEC JTC 1/SC 7), through its Working Group on Software Process Assessment (WG10). SPICE, which stands for “software process improvement and capability determination,” represents a framework for software process assessment, as well as software process improvement. The standard is comparable to CMM, in that it is designed to assure results that are repeatable, objective, and comparable within similar contextual parameters of management, software development process, and product domain. Both ISO and SEI are engaged in efforts to harmonize the CMM models and SPICE. SEI is working to assure that the CMM family of models is consistent with the ISO family of standards.
3.4. Acquisition Strategies

The trend away from standardization of systems engineering process models can be understood better when viewed in the context of system acquisition. A relatively new trend in acquisition management is the widespread use of performance-based contracting (PBC), a procurement tool based on the systems engineering approach that emphasizes the purpose of the acquisition. Figure 16 is based on a systems engineering model of Sage that relates three orthogonal dimensions of systems engineering projects: purpose, structure, and function. By focusing attention on purposeful aspects (goals, needs of the consumer, the “why”), as opposed to the structure (the “what”) and function (the “how”) of the solution, the vendor is not constrained to life cycle models, development tools, methodologies, and products specified by the consumer.

In effect, the consumer and the vendor enter a partnership agreement at a higher level of abstraction than in conventional contracting methods. Figure 16 depicts a relationship among four acquisition strategies:

- in-house development, using contracting strategies such as “level of effort” (LOE);
- conventional strategies, based on “cost plus” contracting;
- PBC; and
- COTS acquisition.

The key determinant in Figure 16 is the level of the agreement between the consumer and the vendor. In the case of in-house development, the needs of the consumer are satisfied through internal means and the consumer manages any externally procured components, personnel, or other resources. The agreement of the consumer with the vendor is at the product level.

In the case of conventional contracting strategies, the consumer and vendor enter into a partnership at the process level of agreement. The consumer—driven by an unsatisfied need—analyzes the problem, synthesizes a solution to the problem, and creates a product specification. The specification is presented to the contractor, who writes a development plan. It is agreement on the vendor’s development plan that is the basis of the transaction between consumer and vendor.

The development plan sets forth a process model (development life cycle), methods, tools, and, in some cases, organizational roles and responsibilities that the vendor agrees to follow. The consumer accepts an oversight role and a responsibility to convey to the vendor enough information from the problem space (the purpose dimension) to insure the successful completion of the solution. However, whenever a problem is reformulated in terms of a solution, information is lost. All the answers to the “why” question have been replaced by a specification. The “lost” information is retained by the consumer organization. Unfortunately, the organization that needs this information is not the consumer, but, rather, the vendor. While this lossy contracting procedure may work for the procurement of small and well-understood systems, large, innovative efforts magnify the loss and improve the probability of failure.

©Encyclopedia of Life Support Systems (EOLSS)
3.5. Concurrent Design

Without a doubt one of the most difficult problems in engineering systems is predicting—or in many cases recognizing—emergent properties of the design, all of which are the consequences of design decisions and some of which may be faults, before the decisions are implemented. Recognition of faults is much more difficult in

![Figure 16. Relationship among four acquisition strategies](image-url)
the early stages of design than in the later stages. At the product level, problems can be seen when failures occur or when validation efforts show that requirements have not been met. At the process level, verification, test, and fault detection may uncover errors, but it is still possible that failure conditions may emerge that are undetected by the development process. The problems are simply much less visible at the process level than at the product level. At the management level, a problem with the design must be very severe to gain recognition in a noisy environment, often dominated by budgetary, scheduling, and political concerns. There are at least two strategies that have been used to ameliorate this difficulty. We refer to these as (a) deferring the commitment (i.e., implementing the design in any form, including documentation) and (b) shortening the learning curve. Figure 17 depicts the situation, characterized by premature design decisions and their documentation, prior to understanding the design and its emergent properties and their impact on the functional characteristics of the product. Figure 17 shows an area, referred to as the “knowledge gap,” that is the result of the practice and the root cause of many engineering failures.

3.5.1. Deferring the Commitment

The first of these two strategies is delaying the commitment until there is adequate knowledge on which to base design decisions. In the traditional waterfall cycle, even with feedback (including spiral and “V” variations), the sequence of “first requirements, then design,” when interpreted as a management tool, suggests that resources allocated to requirements may be different from those used in the design phase—a discontinuity, not in the logic of the model, but in its interpretation. In the late 1990s, many began to recognize that all requirements are design decisions, so the effort needed to produce requirements is design. Requirements are descriptive, not prescriptive. The implication is that high quality requirements cannot possibly be created in a vacuum—there must be a design, the creation of which precedes the writing of requirements and for which the requirements provide a description. Thus, while a system requirements phase logically
precedes a design phase in most acquisition cycles, the order of activities in time follows a design–describe pattern, as shown in Figure 18. These conclusions were influenced by object-oriented software engineering methods that allow the implementation details of classes and objects to be deferred in early development until enough information exists to complete the definitions. In terms of the traditional waterfall life cycle, each of the life cycle phases is composed of one or more turns of the design–describe loop. The number of turns of the loop in any phase is the number needed to complete the phase, which in turn is a qualitative measure of the maturity of the overall system design. Each turn through the design–describe loop adds substance to the design.

Figure 18. The design–describe loop

3.5.2. Shortening the Learning Curve

If the missing element is the ability to see the side effects, or emergent properties, of design decisions, then the software engineering community has many advantages over systems that contain hardware. Even an incomplete software design may be driven rapidly to the product level, including testing, validation, and all other activities. This is because of the degree of automation of software tools, such as programming languages, compilers, and link editors, which allow partial products to be fully built as often as desired. The software build is the capability that allows designs to be tested and emergent properties to be discovered and addressed at the earliest possible moment. A build is much easier to achieve in software, using automated software engineering tools, than it is in systems that contain hardware. For example, if an aircraft—say a new design for Concorde—was the object being developed, then a hardware build would construct a working Concorde, complete in every detail that has been designed at that point in time. Automated support is required if anything approaching a “daily build” is contemplated. The representation must be readily extensible to support new development as well as design modifications. Clearly, no adequate representation technology exists today for creating a “soft” representation of hardware that will support full-scale design and analysis. On the other hand, several “virtual reality” technologies have demonstrated promising results. The automobile and aerospace industries are addressing the need to integrate analysis tools to provide a real time analysis capability.

The life cycle for this activity is dependent on the system architecture. It is highly advantageous to design the system in such a way that early partial products support the design and testing of later partial products. The architecture is analogous to the structure of modern computer operating systems, in which early development provides services that support later, higher layer development. A common visual analogy is the growth of
an onion with its layering logic. One or more layers of the architecture may be packaged as “complete” partial products that may be fully integrated and tested. The goal is to provide a robust baseline on which to predicate further growth and deliver a more complete partial product.

A generic building block for the life cycle is shown in Figure 19. The basic pattern for each period—shown as one full day in the figure—is a concurrent design session in the morning, followed by offline work, as necessary, in the afternoon. Ideally, all work would be accomplished in the concurrent sessions with analysis performed in near-real time. A leader, sometimes referred to as the systems engineer or the conductor, is needed to order the work of the team. To be successful, the leader should be very familiar with the product family from which the product derives, have experience in the identification of the design drivers and a clear idea of the major tradeoff decisions that are important for the design, and have a firm grasp of the analysis processes and tools that will be utilized by the team.

![Figure 19. Generic building block for the life cycle](image)

The activities in the concurrent session are much like the familiar plan–do–check–act cycle of Shewhart and Deming, except that all engineers are working as a unit with a unified model. Analysis tasks that cannot be accomplished during the session are assigned as offline tasks. During the session the concurrent part of the cycle, shown as assign–perform–input–evaluate–improve in Figure 19, is traversed as many times as is comfortable during the allotted time for the concurrent session. Single partial products may span several sessions before they become part of the product baseline. Each baselined partial product is a new layer of the “onion.” Reviews with management and other stakeholders may be scheduled between partial products if needed. Although this building block may be generalized to any design activity in any basic life cycle model, the most powerful application appears to be in new projects or in projects in which visualization is critical for early unification of competing designs.
It has been pointed out that the structure of most life cycles comes from a common idea: that of the waterfall model of Royce and his successors. We see similarities in the concurrent design model to other life cycle models. For example, Boehm’s spiral model captures the essential shape. Most models that are called evolutionary models capture the idea of building stable subsets of products and then layering subsequent development on them. Finally, software build synchronizes the product baselines of all members of a software team. However, the concurrent design process adds the new element of a continuous build, which in its effect shortens the length of the design–describe loop to zero. We therefore use the word “unification” instead of synchronization. Assessment is continuous, since the concurrent sessions, in theory, represent all design elements. In practice, reviews are needed to validate the design of partial products across the widest possible set of interests.

Glossary

Life cycle: A model of the progression of a system from its earliest conceptualization; through its realization as system requirements, design documentation, and physical artifact; continuing throughout its deployment, operation, and termination; and including its transition to a replacement system.

Lossy: A term of data management, which refers to the loss of information that occurs when data are transformed from one schema to another. In general, the term refers to transformations in which some of the meaning is lost. An example is the translation of words between two natural languages.

Process: A continuous change of state of a system, often modeled by discrete steps or phases, and leading toward a particular result. A process model is referred to as a life cycle.

System acquisition: A general term designed to include a number of possible ways of gaining a material interest in a system, such as in-house development, purchase, lease, or any other orderly process. In an organization, system acquisition includes all processes, tools, and personnel involved in instantiating a product or service that is sold in the planning and marketing process.

Systems engineering: The prediction, detection, analysis, and utilization or control of emergent properties of systems. A system is a collection of parts such that analytical knowledge of the parts is insufficient to completely predict the behavior and other characteristics of the whole. Properties of the whole that are the result of interaction of the parts, or that are the result of addressing the collection of parts as a whole, are emergent properties.

Waterfall cycles: A linear, partial ordering of system definition, development, and deployment activities, such that the engineering products of these activities flow from predecessor to successor after the fashion of falling water. Since water does not naturally flow uphill, the model suggests that special treatment is necessary to revisit a predecessor activity, when unexpected properties of the design emerge. The introduction of the waterfall model is widely
attributed to Winston Royce.

Bibliography


Biographical Sketch

**F.G. Patterson Jr.** received a B.A. degree from Rhodes College, two M.S. degrees from the University of Memphis, a Sc.M. degree from Brown University, and a Ph.D. degree from George Mason University: the most recent in 1996. He has been a faculty member at several universities, in departments of computer science and systems engineering; has worked as a systems engineer in the aerospace industry, and has worked for the National Aeronautics and Space Administration (NASA). His experience includes extensive assignments on large, complex systems undertaken by the Federal Aviation Administration and by NASA. His interests include systems engineering and systems management, especially the areas of process improvement, software reuse, and new paradigms for rapid system design.

To cite this chapter