Chapter 2

Software Life Cycle

CONTENTS

This page intentionally left blank.

Chapter 2

Software Life Cycle

2.1 Introduction

In the early days of computing, software was developed by many individuals following their own methods. Often, the methods employed some form of “code and fix”, where the programmer writes some code and then tests it to see how it performs. The programmer then uses the test results to modify or fix the code and tests again. Programmers were able to get by with this type of development for two reasons. First, no better way had been developed, and second, software was not that complex. As software grew more complicated and organizations relied on computers for more of their operations, including finances and even human lives, this laissez faire approach to programming gave way to more disciplined methods. The overall framework in which software is conceived, developed, and maintained is known as the Software Development Life Cycle (SDLC). This chapter discusses the various types of SDLCs, along with their advantages and disadvantages.

A life cycle model defines the phases, milestones, deliverables, and evaluation criteria of the software development process. These form the basis of the work breakdown structure (WBS), used for project planning and management.

2.2 Process Description

Life cycles are usually referred to as models, and define the phases of a software development effort. Simple life cycles may have only three phases, Design, Development, and Maintenance; while complex life cycles may include 20 or more phases. Generally, software life cycles include the phases shown in Figure 2-1.

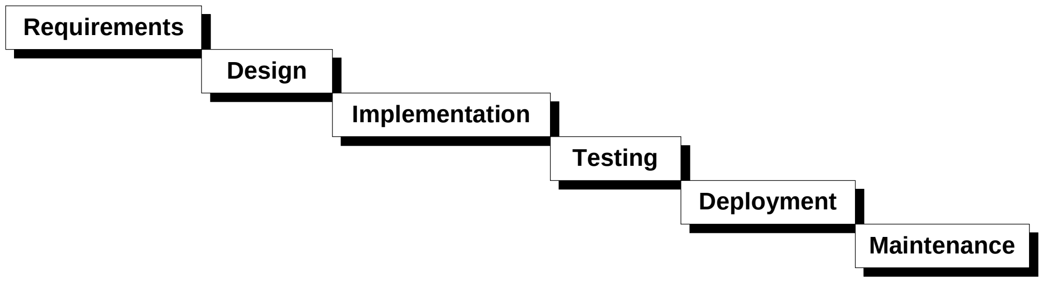

Figure 2-1 Common Life Cycle Phases

These “classic” phases are often divided into additional phases to allow better definition and control of the development process. They may also be repeated in an iterative manner, depending on the software complexity and the life cycle model used. Most life cycle phases are identical or similar to the common phases identified above and the following general descriptions will apply across most models. Note that single phases are composed of multiple activities.

The *Requirements Phas*e consists of analyzing the problem or need for which the software is being developed. This analysis, a systems engineering activity, develops and specifies requirements, stating what the software must do. In addition to stated requirements, requirements are derived from higher-level requirements and statements of need.

In the *Design Phase* the software structure is defined. Technical approaches are chosen and problems are solved conceptually. This phase is often divided into a Preliminary Design Phase and a Detailed Design Phase. In the preliminary design the initial software architecture is developed. In the detailed design, functional modules are defined, along with user interfaces and interfaces between modules.

The *Implementation Phase* (sometimes called the Development Phase) is where the programming or coding takes place to execute the software design. This phase is often iterative, with unit and integration testing being performed after a software build, and the results used in another round of programming.

Software is tested for functionality and requirements compliance during the *Testing Phase*. Testing is often split into three separate phases: Unit Testing, Integration Testing, and Acceptance Testing. The first two may be part of a repeated cycle of coding and testing, while acceptance testing verifies requirements compliance.

During the *Deployment Phase* the software is installed in the intended system and users are trained in its operation. At this point the software development effort is considered complete.

The *Maintenance Phase* includes fixing errors and modifying or upgrading the software to provide additional functionality, such as enabling the software to work with new computing platforms. This phase costs far more in time and effort than the original development. Software maintainers must relearn the original software code to understand what was done, then make changes to specific modules to produce the desired effect without interfering with rest of the software. It's much easier to change requirements earlier than it is to change software code later. This also means that software should be developed with maintenance in mind.

2.2.1 Life cycle Models

There are many life cycle models, or paradigms. Most of them are variations of three classic software development models: the waterfall, incremental, and spiral models. These three, along with the evolutionary model, are summarized here.

2.2.1.1 Waterfall Model

The waterfall model, also known as the linear sequential model, is shown in Figure 2-2 with its major phases, milestones, and products. It is a highly structured development process, first used on DoD software projects in the 1970s. It is the "traditional" approach to software development and was derived from defense and aerospace project lifecycles. It is considered superior to the previously used "code and fix" methods of software development, which lacked formal analysis and design.

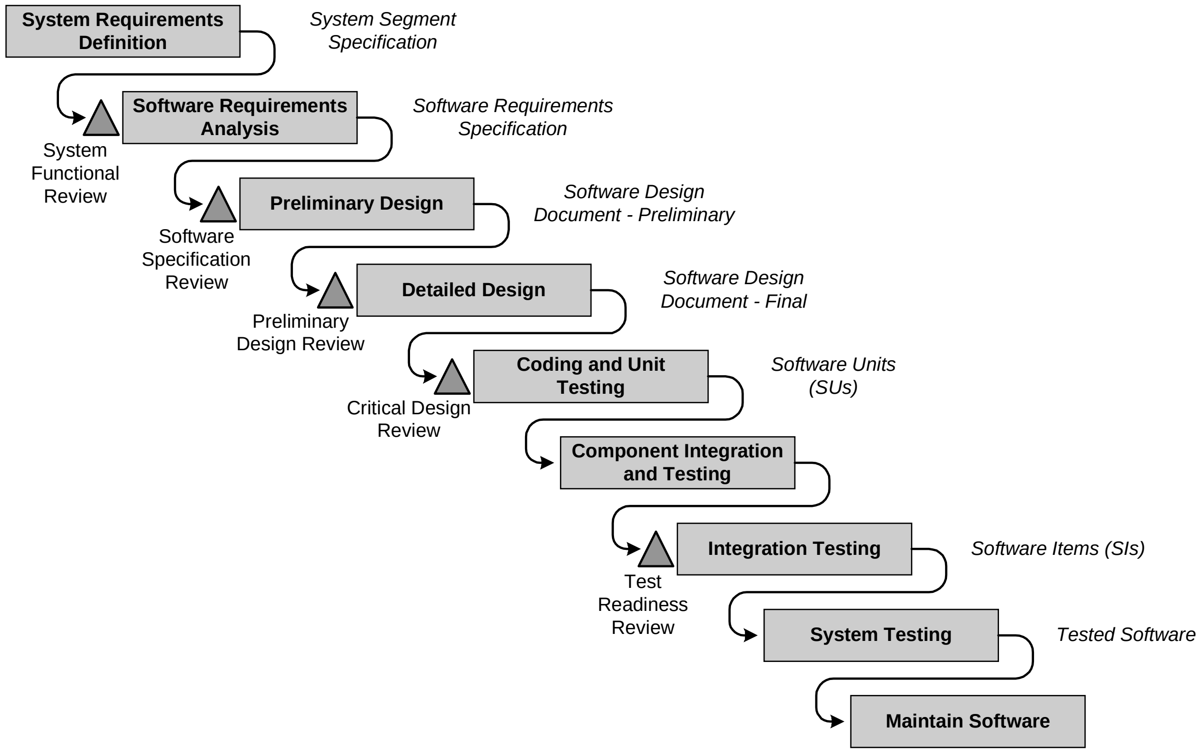
System Requirements Definition
System Segment Specification
Software Requirements Analysis
Software Requirements Specification
System Functional Review
Preliminary Design
Software Design Document - Preliminary
Software Specification Review
Detailed Design
Software Design Document - Final
Preliminary Design Review
Coding and Unit Testing
Software Units (SUs)
Critical Design Review
Component Integration and Testing
Integration Testing
Software Items (SIs)
Test Readiness Review
System Testing
Tested Software
Maintain Software

Figure 2-2 Waterfall Model [1]

The waterfall model is documentation-intensive, with earlier phases documenting *what* must be done and subsequent phases adding greater detail and defining *how* it should be done. The output from one phase serves as the input to the next phase, with the project flowing from one step to the next in a waterfall fashion. Phases are assumed to be sequential, with only localized feedback during the transition between phases. This is accomplished by using reviews as gates. Comprehensive reviews validate the work of one phase, and require the resolution of any problems before development is allowed to proceed to the next phase.

An important consideration for the Waterfall model is that fixes or modifications are often put off until the maintenance phase. This can be very costly, as the cost to correct a problem gets higher with each successive phase.

Advantages

- System is well documented.
- Phases correspond with project management phases.
- Cost and schedule estimates may be lower and more accurate.
- Details can be addressed with more engineering effort if software is large or complex.

Disadvantages

- All risks must be dealt with in a single software development effort.
- Because the model is sequential, there is only local feedback at the transition between phases.
- A working product is not available until late in the project.
- Progress and success are not observable until the later stages. If a mistake or deficiency exists in the documentation of earlier phases, it may not be discovered until the product is delivered.
- Corrections must often wait for the maintenance phase.

Application

The Waterfall model can be successfully used when requirements are well understood in the beginning and are not expected to change or evolve over the life of the project. Project risks should be relatively low.

2.2.1.2 Incremental Model

The incremental model is essentially a series of waterfall cycles. One variant is shown in Figure 2-3. The requirements are known at the beginning of the project and are divided into groups for incremental development. A core set of functions is identified in the first cycle and is built and deployed as the first release. The software development cycle is repeated, with each release adding more functionality until all requirements are met. Each development cycle acts as the maintenance phase for the previous software release. While new requirements that are discovered during the development of a given cycle can be implemented in subsequent cycles, this model assumes that most requirements are known up front. The effort is planned and executed to satisfy the initial list of requirements. A modification to the incremental model allows development cycles to overlap. That is, a subsequent cycle may begin before the previous cycle is complete.

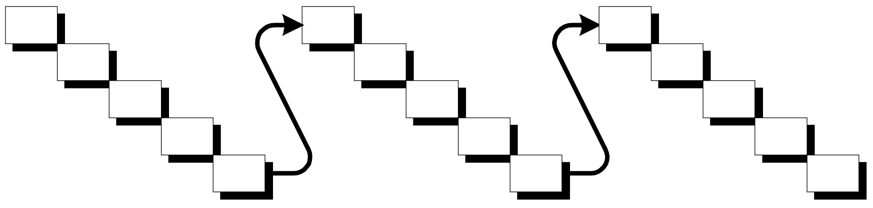

Figure 2-3 The Incremental Model is a Series of Waterfalls

Advantages

- Provides some feedback, allowing later development cycles to learn from previous cycles.
- Requirements are relatively stable and may be better understood with each increment.

- Allows some requirements modification and may allow the addition of new requirements.
- It is more responsive to user needs than the waterfall model.
- A usable product is available with the first release, and each cycle results in greater functionality.
- The project can be stopped any time after the first cycle and leave a working product.
- Risk is spread out over multiple cycles.
- This method can usually be performed with fewer people than the waterfall model.
- Return on investment is visible earlier in the project. [7]
- Project management may be easier for smaller, incremental projects. [7]
- Testing may be easier on smaller portions of the system.

Disadvantages

- The majority of requirements must be known in the beginning.
- Formal reviews may be more difficult to implement on incremental releases than on a complete system. [2]
- Because development is spread out over multiple iterations, interfaces between modules must be well-defined in the beginning. [2]
- Cost and schedule overruns may result in an unfinished system.
- Operations are impacted as each new release is deployed.
- Users are required to learn how to use a new system with each deployment.

Application

The incremental model is good for projects where requirements are known at the beginning, but which need functionality early in the project or which can benefit from the feedback of earlier cycles. Because each cycle produces a working system, it may also be advantageous for projects whose continued funding is not assured and may be cut at any time. It is best used on low to medium-risk programs. If the risks are too high to build a successful system using a single waterfall cycle, spreading the development out over multiple cycles may lower the risks to a more manageable level. [3]

2.2.1.3 Evolutionary Model (Prototyping)

The evolutionary model, like the incremental model, develops a product in multiple cycles. Unlike the incremental model, which simply adds more functionality with each cycle, this model produces a more refined prototype system with each iteration. The process, shown in Figure 2-4, begins in the center with initial requirements and plans, and progresses through multiple cycles of planning, risk analysis, engineering, and customer evaluation. Each cycle produces a prototype that the customer evaluates, followed by a refinement of requirements.

Specification, development, and testing activities are carried out concurrently (in the engineering quadrant) with rapid feedback. Since requirements continue to change, documentation is minimal, although essential information must still be included for understanding the system and for future support. Implementation compromises are often made in order to get the prototype working – permanent fixes can be made with the next prototype. Operational capability is achieved early, but users must be willing to learn how to use each new prototype.

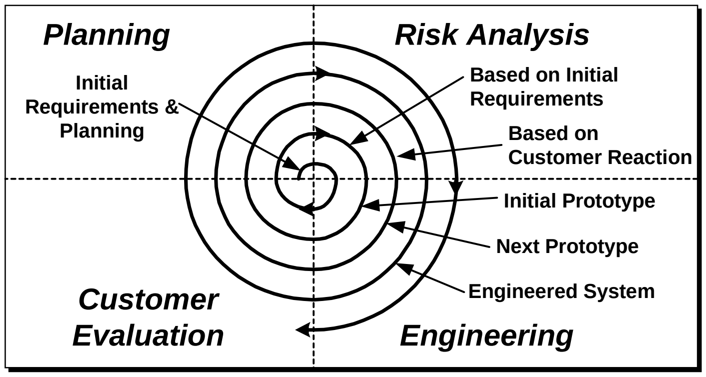

Figure 2-4 First Generation Evolutionary Model [4]

General system requirements must be known prior to development. This is particularly helpful where evolving technology is being introduced into the project. The evolutionary model relies heavily on user feedback after each implementation to refine requirements for the next evolutionary step.

Advantages [5]

- Project can begin without fully defining or understanding requirements.
- Final requirements are improved and more in line with real user needs.
- Risks are spread over multiple software builds and controlled better.
- Operational capability is achieved earlier in the program.
- Newer technology can be incorporated into the system as it becomes available during later prototypes.
- Documentation emphasizes the final product instead of the evolution of the product. [2]
- This method combines a formal specification with an operational prototype. [2]

Disadvantages [5]

- Because there are more activities and changes, there is usually an increase in both cost and schedule over the waterfall method.
- Management activities are increased.
- Instead of a single switch over to a new system, there is an ongoing impact to current operations.
- Configuration management activities are increased.
- Greater coordination of resources is required.
- Users sometimes mistake a prototype for the final system.
- Prototypes change between cycles, adding a learning curve for developers and users.
- Risks may be increased in the following areas:
  - Requirements – Temptation to defer requirements definition.
  - Management – Programs are more difficult to control. Better government/contractor cooperation needed.
  - Approval – Vulnerable to delays in funding approval, which can increase schedule and costs.
  - Architectural – Initial architecture must accommodate later changes.
  - Short term benefits – Risk of becoming driven by operational needs rather than program goals.
  - Risk avoidance – Tendency to defer riskier features until later.
  - Exploitation by suppliers – Government bargaining power may be reduced because initial contract may not complete the entire task, and subsequent contracts are not likely to be competed.
  - Patchwork quilt effects – If changes are poorly controlled, the product quality can be compromised.

Application

The evolutionary model can be employed on most types of acquisitions. However, it is usually employed on medium to high-risk systems. The evolutionary model should be considered for systems where requirements are not all known or not yet refined, but are expected to evolve. It is more applicable to new systems than upgrading existing software. [2] The developing and using organizations must be flexible and willing to work with evolving prototypes. Programs well suited to employ the evolutionary model have some or all of the following characteristics. [5]

- Software intensive systems.
- Have rapidly changing software technology.
- Humans are an integral part of the system.
- Have a large number of diverse users.
- Developing an unprecedented system.
- Limited capability is needed quickly.

2.2.1.4 Spiral Model

The spiral model [6] was developed with the goal of reducing risk in the software life cycle. It combines elements of the waterfall, evolutionary, and incremental models, and depending on how it is implemented can strongly resemble any combination of the others. The model's spiral nature can be seen in Figure 2-5, one of several variants. The project starts at the center and progresses through multiple cycles, each working through the software development activities associated with the four quadrants:

1. Determine objectives, alternatives, constraints.
2. Evaluate alternatives. Identify and resolve risks.
3. Develop the next-level product.
4. Plan the next phase.

Risk management is a key element of the Spiral model and each round of the spiral identifies problems with the highest risk and develops solutions for that set of problems. The process may even resemble a waterfall with additional risk management techniques. Each cycle ends in a review in which stakeholders agree on plans for the next cycle. While a prototype may be produced for IOC, software is usually not developed for release until the last cycle. [7]

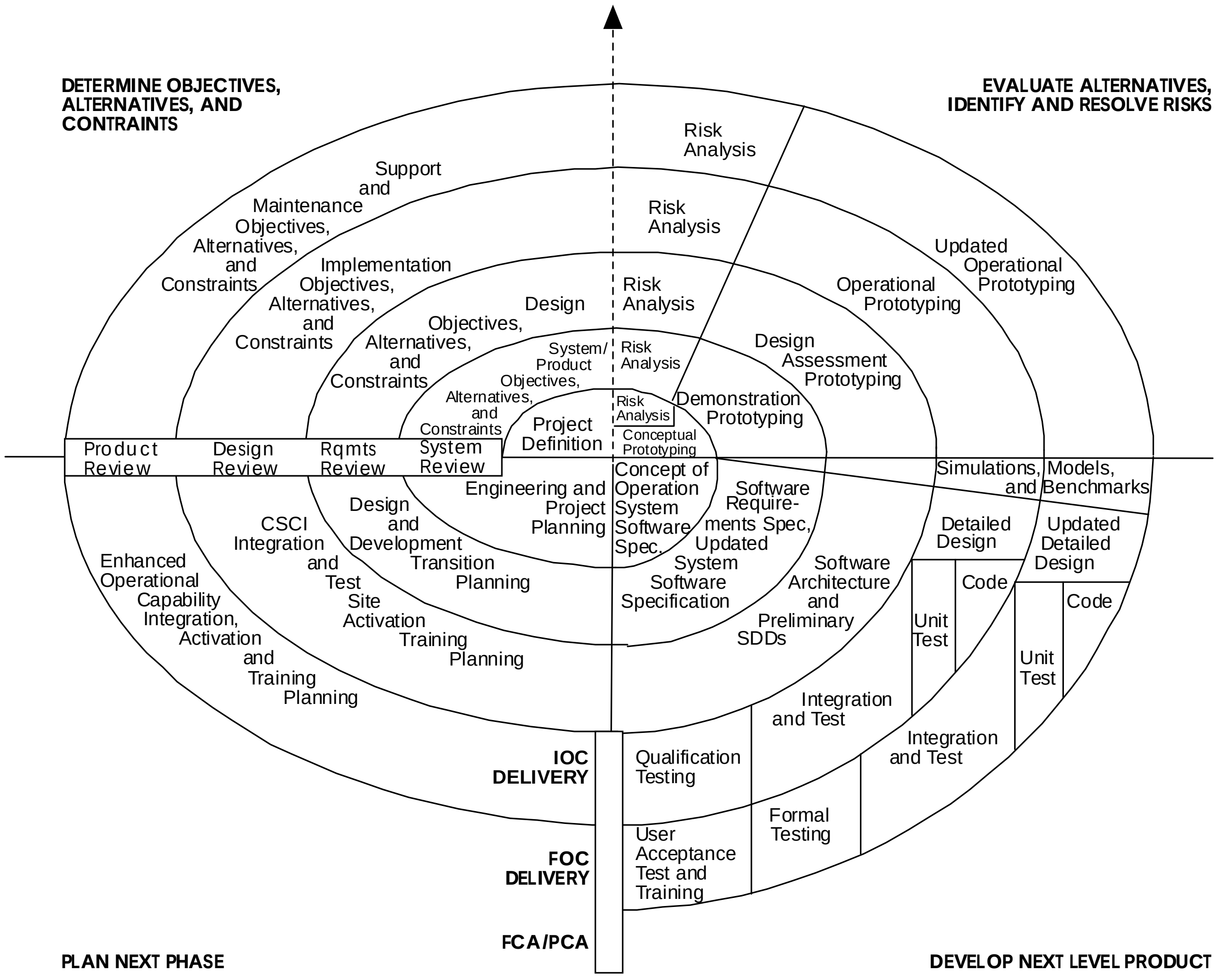
DETERMINE OBJECTIVES, ALTERNATIVES, AND CONTRAINTS
EVALUATE ALTERNATIVES, IDENTIFY AND RESOLVE RISKS
PLAN NEXT PHASE
DEVELOP NEXT LEVEL PRODUCT
Support and Maintenance Objectives, Alternatives, and Constraints
Implementation Objectives, Alternatives, and Constraints
Design Objectives, Alternatives, and Constraints
System/ Product Objectives, Alternatives, and Constraints
Project Definition
Risk Analysis
Risk Analysis
Risk Analysis
Risk Analysis
Risk Analysis
Conceptual Prototyping
Demonstration Prototyping
Design Assessment Prototyping
Operational Prototyping
Updated Operational Prototyping
Product Review
Design Review
Rqmts Review
System Review
Simulations, Models, and Benchmarks
Concept of Operation System Software Spec.
Software Require- ments Spec, Updated System Software Specification
Software Architecture and Preliminary SDDs
Detailed Design
Updated Detailed Design
Code
Code
Unit Test
Unit Test
Integration and Test
Integration and Test
Qualification Testing
Formal Testing
User Acceptance Test and Training
Engineering and Project Planning
Design and Development Transition Planning
CSCI Integration and Test Site Activation Training Planning
Enhanced Operational Capability Integration, Activation and Training Planning
IOC DELIVERY
FOC DELIVERY
FCA/PCA

Figure 2-5 The Spiral Model [5]

The Spiral model has been used extensively in the commercial world because of its performance in a market-driven environment. It significantly reduces technical risk and more easily incorporates new technology and innovations. At the same time, it tends to increase cost and schedule risks. In the past the Spiral model has been difficult to implement in the DoD environment. This is because contracts with predefined deliverables and schedules do not easily accommodate repeating phases, requirement tradeoffs, and changing deliverables. [5] However, use of the Spiral model, where applicable, is directed by DoDI 5000.2, and it has become somewhat popular in the Defense and Aerospace industries.

Advantages

- It provides better risk management than other models.
- Requirements are better defined.
- System is more responsive to user needs.

Disadvantages

- The spiral model is more complex and harder to manage.
- This method usually increases development costs and schedule.

Application

The spiral method should be considered for projects where risks are high, requirements must be refined, and user needs are very important.

2.3 Application

2.3.1 Evolutionary Acquisition and Spiral Development [8]

The Evolutionary Acquisition and Spiral Development approach to acquisition and development has been used to satisfy the preference for evolutionary acquisition strategies established in DoD Directive 5000.1 and DoD Instruction 5000.2. This approach combines incremental, evolutionary, and spiral methods to reduce cycle time and speed delivery of advanced capability to warfighters. It can develop or acquire both hardware and software in manageable pieces and allow the insertion of new technology and capabilities over time.

Evolutionary acquisition fields an initial hardware or software increment (or block) of capability, providing improved capability in a shorter period of time than is possible with a full development effort. This initial deployment is followed by subsequent increments of capability delivered by multiple development cycles. Two variations of this approach are possible. In one, the ultimate capability may be known at the beginning of the program. In the other, the final capability evolves and is defined by matching maturing technologies to the evolving needs of the users.

Spiral development is the iterative process used for developing a defined set of capabilities in a single increment. An increment is a militarily useful and supportable operational capability that can be developed, produced or acquired, deployed, and sustained. An increment may include more than one spiral. Each increment has its own objectives set by the user. An extension of this method can also be applied to Pre-planned Product Improvement (P3I) strategy which adds improved capability to a mature system. This approach is shown in Figure 2-6.

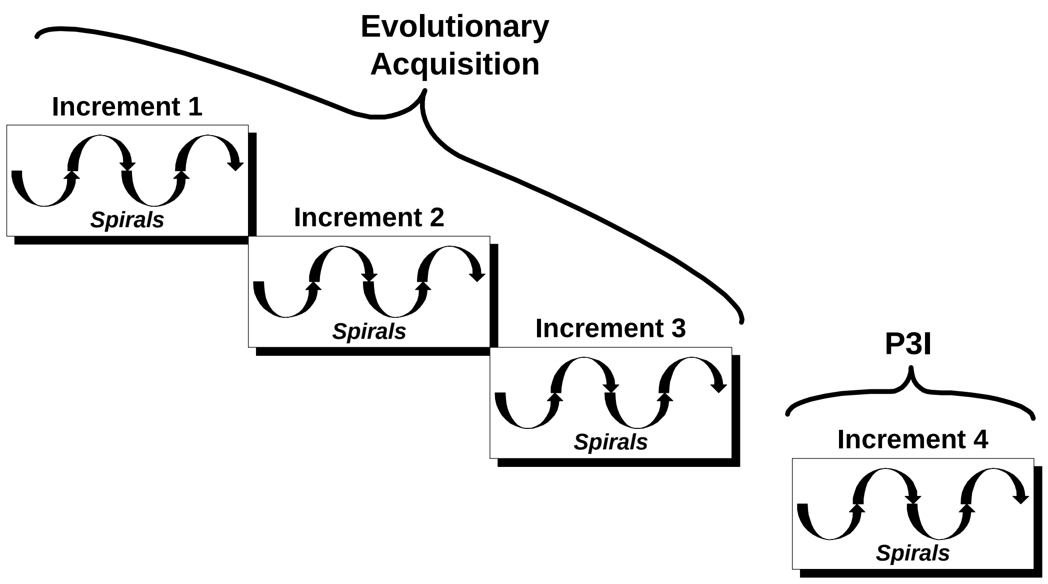

Figure 2-6 Evolutionary Acquisition, Spiral Development, and P3I

2.3.2 Selection Matrix

The purpose of Table 2-1 is to aid in the selection of a life cycle model. It lists various project constraints, along with their applicability to models discussed in this chapter. The applicability is listed for the standard definitions of the models and may not apply equally to modified models. [7] [9]

Table 2-1 Life Cycle Selection Matrix

Legend

Symbol	Meaning
✯	Method is recommended for this constraint
✉	Method is satisfactory for this constraint.
(none)	Method is not recommended for this constraint.

	Waterfall	Incremental	Evolutionary	Spiral
1. Requirements are known and stable.	✯	✯		
2. User needs are unclear/not well defined			✯	✉
3. An early initial operational capability is needed.		✯	✯	
4. Early functionality is needed to refine requirements for subsequent deliveries.		✯	✯	
5. Significant risks need to be addressed.		✉	✯	✯
6. Must interface with other systems.	✉	✉		✉
7. Need to integrate new or future technology.			✯	✯
8. Software is large or complex	✯	✯	✉	✉
9. Software is small or limited in functionality.		✉	✉	✉
10. Software is highly interactive with user.		✉	✉	✉

	Waterfall	Incremental	Evolutionary	Spiral
11. Software involves client/server function.	⌧	⌧	⌧	✯
12. Initial cost and schedule estimates must be followed.	✯	✯		⌧
13. Detailed documentation necessary.	✯	✯		⌧
14. Minimize impact on current operations.	✯			⌧
15. Full system must be implemented.	✯	⌧		⌧
16. Reduce the number of people required.		✯	✯	
17. Project management must be simpler.	✯	⌧		
18. System must be responsive to user needs.		⌧	✯	✯
19. Progress must be demonstrated early.		✯	✯	
20. User feedback is needed.		✯	✯	⌧
21. Reduce the costs of fixes and corrections.		⌧	✯	⌧

2.4 Life Cycle Checklist

This checklist is provided to assist you in choosing an appropriate life cycle for your project if you are beginning a development effort, or to ensure you understand your development life cycle if your project is already under way. If you cannot check an item off as affirmative, you need to rectify the situation yourself or get help in that area. [8]

2.4.1 Beginning a Development Project

- ❑ Do you have an understanding of common life cycle models, along with their strengths, weaknesses, and constraints?
- ❑ Has the operational concept been analyzed to determine what life cycle method would best support the acquisition?
- ❑ Can the requirements be fully defined prior to the beginning of the project? Are they stable?
- ❑ Do you know the timeline for deployment of the new system?
- ❑ Are funds secure for development of the entire system?
- ❑ Are the risks identified?
- ❑ Will risks impact the ability of the project to move forward at crucial points in the system development?
- ❑ Is new or developing technology to be used in the system?
- ❑ Will there be a parallel hardware development effort?
- ❑ Do you understand the level of complexity of the system to be developed?
- ❑ Do you know what the interfaces to existing and future systems are?
- ❑ Do you know the size and magnitude of the development effort?
- ❑ Do you understand the users' needs?
- ❑ Are users able or expected to participate in the development?
- ❑ Do you know what types of acquisition contracts are available for this effort?
- ❑ When choosing a life cycle model, do you know why you are choosing it over other models?

2.4.2 Development Project is Under Way

- ❑ Do you know what life cycle model was selected for your project?
- ❑ Do you understand the project aspects pertaining to the life cycle:
  - ❑ Phases – What are they and what are they supposed to accomplish?
  - ❑ Milestones – What are they? What is their significance?
  - ❑ Criteria for transitioning from one phase to another?
  - ❑ Deliverables – What is expected, during phases, and at the end of the project?
  - ❑ Reviews – What is reviewed when? Who are the reviewers? What actions follow a successful review, an unsuccessful review? What are the entry and exit criteria for each review?
  - ❑ Feedback mechanisms – How is feedback obtained? Who provides it? Who receives it? How is it used?
  - ❑ Documentation – What is to be produced and what it is used for?
- ❑ Do you know where your project is in the life cycle?
- ❑ Is your project following the life cycle?

2.5 Regulations

– Clinger-Cohen Act of 1996, The National Defense Authorization Act for Fiscal Year 1996.

– DoD 5000.2-R, *Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information System Acquisition Programs*, June 2001.

– FAR -- Part 39; *Acquisition of Information Technology*; (FAC 97-27); 25 June 2001.

2.6 References

[1] Brundick, Bill, Editor, *Parametric Cost Estimating Handbook*, Chapter 5, Naval Sea Systems Command, Fall 1995.

[2] Sorensen, Reed, “A Comparison of Software Development Methodologies,” *Crosstalk*, January 1995

[3] Whitgift, David, Methods and Tools for Software Configuration Management, 1991, p17.

[4] Pressman, Roger S., “Understanding Software Engineering Practices, Required at SEI Level 2 Process Maturity,” Software Engineering Training Series, Software Engineering Process Group, 30 July 1993.

[5] *Guidelines for the Successful Acquisition and Management of Software Intensive Systems* (GSAM), Version 3, Chapter 5, USAF Software Technology Support Center, May 2000.

[6] Boehm, Barry W., “A Spiral Model of Software Development and Enhancement,” IEEE Computer, May 1988.

[7] McKenzie, Charlotte A., MIS327 - Systems Analysis and Design, Course Schedule, 1999.

[8] Memorandum from the Undersecretary of Defense for Acquisition Technology and Logistics, Subject: Evolutionary Acquisition and Spiral Development, 12 April 2002.

[9] Quann, Eileen, personal communication to Lloyd K. Mosemann, II, September 1995.

2.7 Resources

2.7.1 Software Life Cycle

Association for Computing Machinery (ACM), ISO 12207 and Related Software Life-Cycle Standards:
www.acm.org/tsc/lifecycle.html

Crosstalk magazine articles:

– “Comparison of Software Development Methodologies”
www.stsc.hill.af.mil/crosstalk/1995/jan/comparis.asp

- “Spiral Model as a Tool for Evolutionary Acquisition” www.stsc.hill.af.mil/crosstalk/2001/may/boehm.asp
- “Prototypes: Tools That Can Be Used and Misused” www.stsc.hill.af.mil/crosstalk/1994/jan/xt94d01g.asp
- “Extreme Methodologies for an Extreme World” www.stsc.hill.af.mil/crosstalk/2001/jun/leishman.asp
- “Dispelling the Process Myth: Having a Process Does Not Mean Sacrificing Agility or Creativity” www.stsc.hill.af.mil/crosstalk/2001/nov/glazer.asp
- “Balancing Discipline and Flexibility With the Spiral Model and MBASE” www.stsc.hill.af.mil/crosstalk/2001/dec/boehm.asp
- “Customizing the Software Process to Support Avionics Systems Enhancements” www.stsc.hill.af.mil/crosstalk/2001/sep/donzelli.asp
- “Tailoring a Software Process for Software Project Plans Part 1: Context and Making Tailoring Decisions” www.stsc.hill.af.mil/crosstalk/1996/apr/tailorin.asp
- “Tailoring a Software Process for Software Project Plans Part 2: Documenting the Project's Defined Software Process” www.stsc.hill.af.mil/crosstalk/1996/may/tailorin.asp
- “MIL-STD-498: What's New and Some Real Lessons Learned” www.stsc.hill.af.mil/crosstalk/1996/mar/lesslear.asp

DeGrace, Peter and Stahl, Leslie, *Wicked Problems, Righteous Solutions: A Catalogue of Modern Software engineering Paradigms*, Yourdon Press.

Department of Energy (DOE) *Software Engineering Methodology*, Chapter 2: http://cio.doe.gov/sqse/sem_toc.htm

Department of Justice Systems Development Life Cycle Guidance Document: www.usdoj.gov/jmd/irm/lifecycle/table.htm

Georgia Tech slides of software life cycles: www.cc.gatech.edu/computing/SW_Eng/people/Faculty/Colin.Potts/Courses/3302/1-08-mgt/

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 5, OO-ALC/TISE, May 2000. Available for download at: www.stsc.hill.af.mil/gsam/guid.asp

LevelA Software, Introduction to life cycle models: www.levela.com/software_life_cycles_swdoc.htm

McKenzie, Charlotte A., MIS327 - Systems Analysis and Design, Course Schedule, 1999. See lecture with model selection guide: http://metr.cua.edu/faculty/mckenzie/mis327/MIS%20327%20Course%20Schedule.html

OSD Program Manager’s Guide for Managing Software, Chapter 4 – Life Cycle Models: http://www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Parametric Estimating Handbook, Chapter 6, International Society of Parametric Analysts (ISPA) – Commercial Software Models: www.ispa-cost.org/PEIWeb/ch6.htm

Parametric Cost Estimating Handbook, Chapter 5, DoD, NASA, etc.: www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm

Vision 7, overview of life cycle phases: http://www.vision7.com/SoftwareDevelopment/ProjectLifeCycle/default.asp

2.7.2 Software Acquisition/Development

Air Force Acquisition Gameboard. Must have a .mil web presence: https://afkm.wpafb.af.mil/ASPs/Gameboard

Air Force Publication Web Site – The official source site for Air Force Administration Publications and Forms: http://afpubs.hq.af.mil

Air Force Software Technology Support Center (STSC) documentation and standards: http://stsc.hill.af.mil/doc/index.asp

Best Manufacturing Practices (BMP), TRIMS Risk Management and Best Practices software downloads: www.bmpcoe.org/pmws/index.html Download KnowHow software and copies of DOD 5000.1, 5000.2, etc: www.bmpcoe.org/pmws/download/knowhow.html

Crosstalk magazine. Extensive database of articles on software engineering. Published by the Air force Software Technology Support Center (STSC): http://www.stsc.hill.af.mil/crosstalk/

Defense Acquisition Deskbook. www.deskbook.osd.mil

Defense Acquisition University web site: www.dau.mil

Department of Defense single stock point of military specifications, standards, and related publications: http://dodssp.daps.mil/

DoD Software Information Clearinghouse: www.dacs.dtic.mil

OSD acquisition web site: www.acq.osd.mil

RSPA. R. S. Pressman & Associates library of software engineering knowledge. In particular, see "An Adaptable Process Model": http://www.rspa.com/spi

Software Program Managers Network (SPMN). Sponsored by the Deputy Under Secretary of Defense for Science and Technology (DUSD (S&T), Software Intensive Systems Directorate. www.spmn.com

SPMN Guidebooks available for download at: www.spmn.com/products_guidebooks.html

University of Southern California (USC) Center for Software Engineering (CSE), Technical Reports: http://sunset.usc.edu/publications/TECHRPTS/index.html