Software Engineering Methodology

Chapter 2.0
Lifecycle Model

Table of Contents

Chapter **Page**

Chapter: **2.0**
Lifecycle Model

Description: This chapter describes the lifecycle model used for the Departmental software engineering methodology. This model partitions the software engineering lifecycle into eight major stages, as shown in *Exhibit 2.0-1, Software Lifecycle Stages and Deliverables.* Each stage is divided into activities and tasks, and has a measurable end point (Stage Exit). The execution of all eight stages is based on the premise that the quality and success of a software product depends on a feasible concept, highly visible project planning, commitments to resources and schedules, complete and accurate requirements, a sound design, consistent and maintainable programming techniques, and a comprehensive testing program. The lifecycle stages and activities are described in chapters 3.0 through 10.0.

Intermediate work products are produced during the performance of the activities and tasks in each stage. These work products are inspected and can be used to assess software integrity, quality, and project status. As a result, adequacy of requirements, correctness of designs, and quality of software products become known early in the effort.

At least one time during each stage, an In-Stage Assessment is performed. An In-Stage assessment is an independent review of the work products and deliverables developed or revised during each lifecycle stage. The assessment is typically conducted by a Quality Assurance practitioner and the results are provided to the project manager. In-Stage Assessments are recommended after the achievement of all major project milestones and the completion of deliverable work products.

At the conclusion of each stage, a Stage Exit is initiated to review the work products of that stage and to determine whether to proceed to the next stage, continue work in the current stage, or abandon the project. The approval of the system owner and other project stakeholders at the conclusion of each stage enables both the system owner and the project manager to remain in control of the project throughout its life, and prevents the project from proceeding beyond authorized milestones.

The end products of the lifecycle are the software product, the data managed by the software, associated technical documentation, and user training and support. The end products and services are maintained throughout the remainder of the lifecycle in accordance with documented configuration management procedures.

Description, continued:

The lifecycle model provides a method for performing the individual activities and tasks within an overall project framework. The stages and activities are designed to follow each other in an integrated fashion. Project teams have the flexibility to adapt the lifecycle model to accommodate a particular development methodology (e.g., spiral development), software engineering technique (e.g., prototyping and rapid application development), or other project constraints.

The amount of project and system documentation required throughout the lifecycle depends on the size and scope of the project. System documentation needs to be at a level that allows for full system operability, usability, and maintainability. Typically, projects that require at least one work-year of effort should have a full complement of documentation. For projects that require less than one work-year of effort, the project manager and system owner should determine the documentation requirements. In addition, the project's security and quality assurance criteria may require the performance of other activities and the generation of additional documentation.

The requirements for documentation should not be interpreted as mandating formal, standalone, printed documents in all cases. Progressive documents that continuously revise and expand existing documentation, online documents, forms, reports, electronic mail messages, and handwritten notes (e.g., informal conference records) are some examples of alternative documentation formats.

The following sections provide additional information about the lifecycle model.

Exhibit 2.0-1. Software Lifecycle Stages and Deliverables

Planning	Feasibility Statement Project Plan Software Quality Assurance Plan
Requirements Definition	Software Configuration Management Plan Continuity of Operations Statement/Plan Software Requirements Specification Project Test Plan Acceptance Test Plan (draft)
Functional Design	Logical Model Data Dictionary Requirements Traceability Matrix Functional Design Document
System Design	Physical Model Integration Test Plan (draft) System Test Plan (draft) Conversion Plan System Design Document Program Specifications Programming Standards
Programming	Acquisition Plan Installation Plan (draft) Integration Test Plan (final) System Test Plan (final) Software Baseline Transition Plan Operating Documents (draft) Training Plan (draft)
Software Integration & Testing	Integration Test Reports System Test Report Operating Documents (final) Training Plan (final) Installation Plan (final) Acceptance Test Plan (final) Preacceptance Checklist

Installation & Acceptance	User Training Materials Acceptance Test Report Acceptance Checklist Operational System
Software Maintenance	

Section: **2.1 Project Sizes**

Description: The lifecycle model used in this software engineering methodology can be applied to software projects of varying sizes. In this model, software projects are divided into three sizes: large, medium, and small. Each project size uses the same lifecycle stages. Medium and small projects may compress or combine stages and required documentation in direct proportion to the size of the development effort. The major differences between project sizes are determined by the following items.

- The estimated total labor hours (the level of effort) required to complete the project.
- The use of cutting edge or existing technology.
- The type and extent of both user and system interface requirements.
- The project's contribution to, and impact on, the activities carried out by the system users and other Departmental organizations.

The requirements, constraints, and risks associated with the project also influence the determination of project size. The project size and any plans for adapting the lifecycle model are documented in the Project Plan, which is reviewed and approved by the system and other project stakeholders.

The following subsections provide descriptions of the three project sizes used in this lifecycle model. *Exhibit 2.1-1, Software Project Sizes,* shows the level of effort and complexity measures used to define the three sizes.

Large Projects: Large software engineering projects are included in the system owner's organizational long-range plans. Headquarters-wide and Departmentwide projects are usually developed as large-sized projects and are likely to require a major acquisition of hardware and software. Typically, the larger the size and scope of the project, the greater the detail and coordination needed to manage the project. As risk factors and levels of effort increase, the scope of project management also increases and becomes a critical factor in the success of the project.

Medium Projects: Medium software engineering projects require less effort than large projects, typically use existing hardware and software, and might not be captured during the organizational long-range planning process. Medium size projects are frequently developed to automate operations within a programmatic office or among a limited number of sites, and may be used to interface with other software products. Planning medium size projects within the context of the system owner organization's overall mission, and building in compatibility to the Departmental computing environment can improve the software product's ability to interface with other users, organizations, and applications; and increase the product's longevity.

Small Projects: Small software engineering projects require minimal effort and use existing hardware and software. The operational details of a small project can easily be managed by the project manager, so formal documentation requirements are limited. A project is small when the software being developed will have limited functionality and use, meets a one-time requirement, or is developed using reusable code.

Exhibit 2.1-1. Software Project Sizes

Complexity (and associated characteristics)	**Effort Required** (in staff months)		
	0-8	**9-24**	**25-n**
Low: - Existing or known technology - Simple interfaces - Requirements well known - Skills are available	Small	Small	Medium
Medium: - Some new technology - Multiple interfaces - Requirements not well known - Skills not readily available	Small	Medium	Large
High: - New technology - Numerous complex interfaces - Numerous resources required - Skills must be acquired	Medium	Large	Large

Section: **2.2**
Adapting the Lifecycle

Description: The software engineering methodology implements well-defined processes in a lifecycle model that can be adapted to meet the specific requirements or constraints of any software project. This section provides guidelines for adapting the lifecycle processes to fit the characteristics of the project. These guidelines help ensure that there is a common basis across all software projects for planning, implementing, tracking, and assuring the quality of the work products.

The lifecycle model has built-in flexibility. All of the stages and activities can be adapted to any size and scope software engineering project. The lifecycle can be successfully applied to software development projects, software maintenance or enhancements, and customization of commercial software. The lifecycle is appropriate for all types of administrative, business, manufacturing, laboratory, scientific, and technical applications. For scientific and technical projects, adaptations to the lifecycle may be dictated by the project stakeholders or the requirements for reporting technical results in formal reports or journal articles.

Adaptations: The lifecycle can be compressed to satisfy the needs of a small project, expanded to include additional activities or work products for a large or complex project, or supplemented to accommodate security requirements. Any modifications to the lifecycle should be consistent with the established activities, documentation, and quality standards included in the methodology. Project teams are encouraged to adapt the lifecycle as long as the fundamental software engineering objectives are retained and quality is not compromised.

The following are some examples of lifecycle adaptations.

- Change the order in which lifecycle stages are performed.
- Schedule stages and activities in concurrent or sequential order.
- Repeat, merge, or eliminate stages, activities, or work products.
- Include additional activities, tasks, or work products in a stage.
- Change the sequence or implementation of lifecycle activities.
- Change the development schedule of the work products.
- Combine or expand activities and the timing of their execution.

Adaptations, continued:

The lifecycle forms the foundation for project planning, scheduling, risk management, and estimation. When a lifecycle stage, activity, or work product is adapted, the change must be identified, described, and justified in the Project Plan. *Exhibit 2.2-1, Adapting the Lifecycle,* shows how stages can be combined to accommodate different size projects and software engineering techniques. *Notes* are provided throughout the lifecycle stage chapters to identify activities that have built-in project adaptation strategies. Adaptations should not introduce an unacceptable level of risk and require the approval of the system owner and other project stakeholders.

When adapting the lifecycle model, care must be taken to avoid the following pitfalls.

- Incomplete and inadequate project planning.
- Incomplete and inadequate definition of project objectives and software requirements.
- Lack of a development methodology that is supported by software engineering preferred practices and tools.
- Insufficient time allocated to complete design before coding is started.
- Not defining and meeting criteria for completing one software lifecycle stage before beginning the next.
- Compressing or eliminating testing activities to maintain an unrealistic schedule.

Sample Statements:

The following are sample statements that can be used in the Project Plan to describe different types of lifecycle adaptations. The first example shows a scenario where the Feasibility Study activity will not be conducted in the Planning Stage.

A Feasibility Study will not be performed for this software project. The need for the product has been documented in several organizational reports and was included in the fiscal year long-range plans. The platform for the project is currently used for all applications owned by this organization. There are no known vendor packages that will satisfy the functional requirements described by the system owner.

Sample

Statements, continued:

The following is a sample statement that shows how work products from two different stages can be combined into one deliverable.

The Functional Design and System Design documents will be combined into one design document. A Stage Exit will be conducted when the design document is completed. To reduce the risk associated with combining the two documents, the project team will develop prototype screens and reports for review and approval by the system owner/user(s) as the prototypes are developed.

The following is a sample that shows how the eight lifecycle stages can be compressed into five stages for a small project.

This project will require 4 staff months of effort to enhance an existing application. The eight stages in the lifecycle will be combined into five stages as follows: (1) Planning, (2) Requirements and Design, (3) Programming and Testing, (4) Installation and Acceptance, and (5) Maintenance.

The following deviations will occur for document deliverables:

- *A Feasibility Study and an Analysis of Benefits and Costs will not be necessary due to the restricted software and hardware platform.*
- *The Requirements Specification will be limited to the statement of enhancement requirements.*
- *The Functional Design and System Design documents will be combined into one design document.*
- *An amendment package will be developed for the existing Users Manual.*

Exhibit 2.2-1. Adapting the Lifecycle

```
   PROJECT
     SIZE
                                ▼        ▼        ▼         ▼        ▼         ▼        ▼
                      *        *        *        *         *        *         *        *
 LARGE                /))))))))3))))))))3))))))))3))))))))3))))))))3))))))))3))))))))1
                      Maintenance
                      *Planning *Req.Def. *Fun.Des. *Sys.Des. *Progrmg. *Testing  *Install. * &
Ops.
                                                     +))))))))))))))),          Accpt.
                                                     *ITERATIVE DEV. /)),(1)
                                                  +))3))))))))))))))1◂)-
                                                  .)▸*next function  *
                                                     .))))))))))))))-

                                ▼              ▼                  ▼              ▼

                      *        *              *                  *              *
 MEDIUM               /))))))))3)))))))))))))3)))))))))))))))3))))))0))))))1  Maintenance
                      *Planning *Req./Fun.Des. *Sys.Des./Progrm. *Test.  *Instl.* & Ops.
                                 +))))))))))))),                      Accpt.
                              +)▸*RAPID PROTO. /)),(2)
                              .))2))))))))))))-◂)-

                                             ▼                        ▼

                      *                      *                        *
 SMALL                /))))))))0)))))))))))3))))))))))))0)))))))))1  Maintenance
                      *Planning *Req./Desg.  *Progr./Test.  *Install.  * & Ops.
                      R         R            R              RAccept.   R
```

LESS))Q ▸
MORE

DEGREE OF PROJECT MANAGEMENT REQUIRED

Note: Iterative development and rapid prototyping are optional techniques that can be used on any size project.

▾ = Stage exit occurs at this point.

(1) Each iteration produces working function(s) from integrated program modules.
(2) May produce any or all of requirements, system architecture, system design.

Section: **2.3**
Development Methodologies

Description: This section describes some examples of development methodologies and techniques that can be used with the Departmental software engineering methodology. The examples include high-level instructions on how to adapt the lifecycle stages to accommodate the development methodology. *Exhibit 2.2-1, Adapting the Lifecycle,* shows how some development methodologies and techniques can be used with the lifecycle model. The examples provided here are not intended to be a comprehensive list of development methodologies and techniques.

Segmented Development: Segmented development is most often applied to large software engineering projects where the project requirements can be divided into functional segments. Each segment becomes a separate project and provides a useful subset of the total capabilities of the full product. This segmentation serves two purposes: to break a large development effort into manageable pieces for easier project management and control; and to provide intermediate work products that form the building blocks for the complete product.

The lifecycle processes and activities are applied to each segment. The overall system and software objectives are defined, the system architecture is selected for the overall project, and a Project Plan for development of the first segment is written and approved by the system owner.

Segments are delivered to the system owner for evaluation or actual operation. The results of the evaluation or operation are then used to refine the content of the next segment. The next segment provides additional capabilities. This process is repeated until the entire software product has been developed. If significant problems are encountered with a segment, it may be necessary to reexamine and revise the project objectives, modify the system architecture, update the overall schedule, or change how the segments are divided.

Two major advantages of this approach are: the project manager can demonstrate concrete evidence that the final product will work as specified; and users will have access to, and use of, segments or functions prior to the delivery of the entire software product.

Spiral

Development: Spiral development repeats the planning, requirements, and functional design stages in a succession of cycles in which the project's objectives are clarified, alternatives are defined, risks and constraints are identified, and a prototype is constructed. The prototype is evaluated and the next cycle is planned.

The project objectives, alternatives, constraints, and risks are refined based on this evaluation; then, an improved prototype is constructed. This process of refinement and prototyping is repeated as many times as necessary to provide an incrementally firm foundation on which to proceed with the project.

The lifecycle activities for the Planning, Requirements Definition, and Functional Design Stages are repeated in each cycle. Once the design is firm, the lifecycle stages for System Design, Programming, and Integration and Testing are followed to produce the final software product.

Rapid Prototyping: Rapid prototyping can be applied to any software development methodology (e.g., segmented, spiral). Rapid prototyping is recommended for software development that is based on a new technology or evolutionary requirements.

With the rapid prototyping technique, the most important and critical software requirements are defined based on current knowledge and experience. A quick design addressing those requirements is prepared, and a prototype is coded and tested. The purpose of the prototype is to gain preliminary information about the total requirements and confidence in the correctness of the design approach. Characteristics needed in the final software product, such as efficiency, maintainability, capacity, and adaptability might be ignored in the prototype.

The prototype is evaluated, preferably with extensive user participation, to refine the initial requirements and design. After confidence in the requirements and design approach is achieved, the final software is developed. The prototype might be discarded, or a portion of it used to develop the final product.

The normal software engineering documentation requirements are usually postponed with prototyping efforts. Typically, the project team, project stakeholders, and system owner agree that the prototype will be replaced with the actual software product and required support documentation after proof of the model. The software that replaces the prototype should be developed using the lifecycle processes and activities.

Iterative Technique:

The iterative technique is normally used to develop software products piece by piece. Once the system architecture and functional or conceptual design are defined and approved, system functionality can be divided into logically related pieces called "drivers."

In iterative fashion, the project team performs system design, code, unit test, and integration test activities for each driver, thereby delivering a working function of the product. These working functions or pieces of the software product are designed to fit together as they are developed. This technique allows functions to be delivered incrementally for testing so that they can work in parallel with the project team. It also enables other functional areas, such as documentation and training, to begin performing their activities earlier and in a more parallel effort. In addition, the iterative technique enables progress to be visible earlier, and problems to be contained to a smaller scope.

With each iterative step of the development effort, the project team performs the lifecycle processes and activities.