The Software Development Life Cycle (SDLC)

For Database Applications

Document ID: SDLC
Version: 1.1c

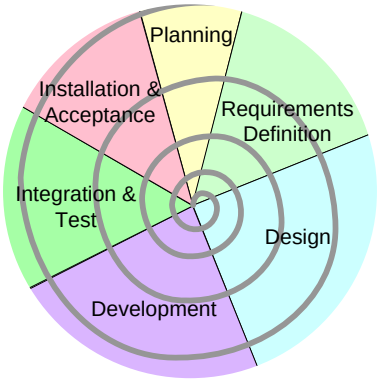


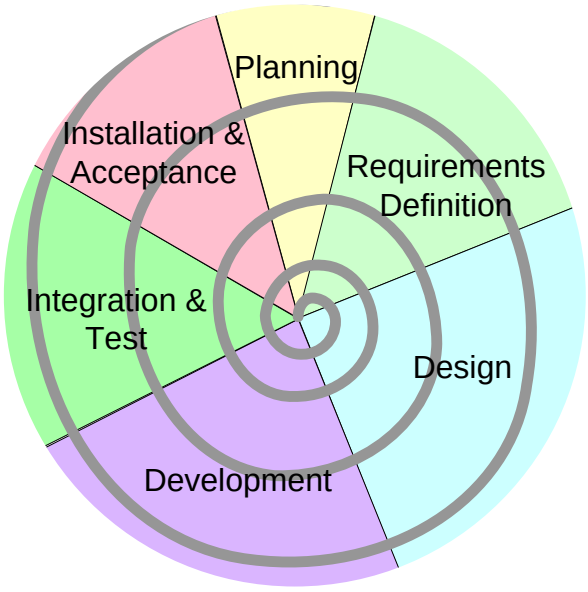
TABLE OF CONTENTS

INTRODUCTION

This document describes the Shell Method™ Software Development LifeCycle (SDLC) for database application development efforts. This chapter presents an overview of the SDLC and associated references. The following chapter describes the internal processes that are common across all stages of the SDLC and the third chapter describes the inputs, outputs, and processes of each stage. Finally, the conclusion describes the four core concepts that form the basis of this SDLC.

ITERATIVE LIFECYCLE

This project uses an iterative development lifecycle, where components of the application are developed through a series of tight iterations. The first iterations focus on very basic functionality, with subsequent iterations adding new functionality to the previous work and/or correcting errors identified for components in production.



The six stages of the SDLC are designed to build on one another, taking the outputs from the previous stage, adding additional effort, and producing results that leverage the previous effort and are directly traceable to the previous stages. During each stage, additional information is gathered or developed, combined with the inputs, and used to produce the stage deliverables. It is important to note that the additional information is restricted in scope; "new ideas" that would take the project in directions not anticipated by the initial set of high-level requirements are not incorporated into the project. Rather, ideas for new capabilities or features that are out-of-scope are preserved for later consideration.

Each iteration through the SDLC produces a small, but critical amount of functionality (or enhancements to existing functionality) in a short time frame (generally six to eight weeks). By focusing efforts into a series of tightly scoped iterations, the project minimizes unwarranted scope creep and optimizes staff effort onto manageable, bite-sized project chunks. After each iteration is completed, the development team generates a list of recommendations for enhancements to be addressed in the next iteration.

Too many software development efforts go awry when the development team and customer personnel get caught up in the possibilities of automation. Instead of focusing on high priority features, the team can become mired in a sea of "nice to have" features that are not essential to solve the problem, but in themselves are highly attractive. This is the root cause of a large percentage of failed and/or abandoned development efforts and is the primary reason the development team utilizes the iterative model.

ROLES AND RESPONSIBILITIES OF PDR AND PER

The iterative lifecycle specifies two critical roles that act together to clearly communicate project issues and concepts between the end-user community and the development team.

PRIMARY END-USER REPRESENTATIVE (PER)

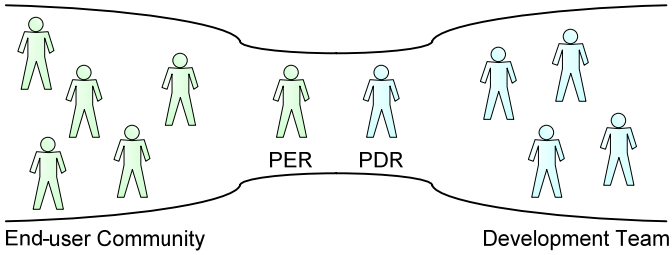
The PER is a person who acts as the primary point of contact and principal approver for the end-user community. The PER is also responsible for ensuring that appropriate subject matter experts conduct end-user reviews in a timely manner.

PRIMARY DEVELOPER REPRESENTATIVE (PDR)

The PDR is a person who acts as the primary point of contact and principal approver for the development community. The PDR is also responsible for ensuring that appropriate technical experts conduct technical reviews in a timely manner.

PER-PDR RELATIONSHIP

The PER and PDR are the "brain trust" for the development effort. The PER has the skills and domain knowledge necessary to understand the issues associated with the business processes to be supported by the application and has a close working relationship with the other members of the end-user community. The PDR has the same advantages regarding the application development process and the other members of the development team. Together, they act as the concentration points for knowledge about the application to be developed.



The objective of this approach is to create the close relationship that is characteristic of a software project with one developer and one end-user. In essence, this approach takes the "pair programming" concept from Agile methodologies and extends it to the end-user community. While it is difficult to create close relationships between the diverse members of an end-user community and a software development team, it is much simpler to create a close relationship between the lead representatives for each group.

When multiple end-users are placed into a relationship with multiple members of a development team, communications between the two groups degrades as the number of participants grows. In this model, members of the end-user community may communicate with members of the development team as needed, but it is the responsibility of all participants to keep the PER and PDR apprised of the communications. For example, this allows the PER and PDR to resolve conflicts that arise when two different end-users communicate different requirements for the same application feature to different members of the development team.

TRAINING AND QUALIFICATION

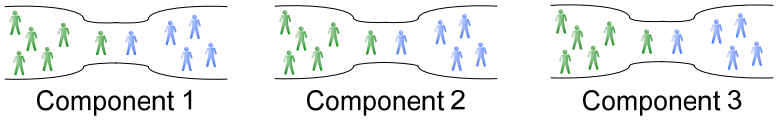
The Project Team Training and Qualification (PTTQ) plan further describes the roles of the PER and PDR, as well as other personnel supporting the software development lifecycle. For each role, the PTTQ provides a description of the activities supported by the role as well as the responsibilities associated with the role. The PTTQ also describes the required training necessary for successful performance of each role, as well as the method of qualification for each role.

Finally, the PTTQ describes training resource requirements and training materials made available to project team members.

COMPONENT-BASED DEVELOPMENT

A key factor in the success of a software development project is the engagement of the end-user community with the development team. Successful projects exhibit a pattern of active participation and quick feedback between the two parties. To accomplish this, it is important that the members of the end-user community have domain-specific knowledge of the business processes to be supported by the application.

In larger software development efforts, it is difficult or impossible for individual members of the end-user community to have full knowledge of all the business processes. For example, end-users with domain knowledge covering marketing and customer relationship management rarely have equivalent knowledge coverage of product inventory and supply chain management. When this happens, it becomes difficult for a single end-user community to adequately specify the functionality and structure of the application. Splitting the application into separate components, each with a dedicated end user community, allows the project to maintain high quality knowledge exchanges by relying on different sets of subject matter experts.



With a component-based development project, each component would have a separate PER-PDR pair, with a separate end-user community for each component. In some cases, the same PDR and development team performs the same function for multiple components. In other cases, the project has separate development teams as well as end-user communities. Finally, the same end-user community and development team may work across more than one or all, components. In this case, the project is broken into components primarily to segregate the development effort and split the documentation stream into smaller, more manageable sets. A lead PER and PDR are identified to coordinate the different sub-teams.

The component-based option for project structure is handled by inserting a Component Iteration Plan (CIP) between the SPMP and the downstream component documentation (requirements, design, etc.). The CIP is further defined in the Planning Stage section of this document.

PROTOTYPES

The software development team, to clarify requirements and/or design elements, may generate mockups and prototypes of screens, reports, and processes. Although some of the prototypes may appear to be very substantial, they're generally similar to a movie set: everything looks good from the front but there's nothing in the back.

When a prototype is generated, the developer produces the minimum amount of code necessary to clarify the requirements or design elements under consideration. No effort is made to comply with coding standards, provide robust error management, or integrate with other database tables or modules. As a result, it is generally more expensive to retrofit a prototype with the necessary elements to produce a production component then it is to develop the component from scratch using the final system design document.

For these reasons, prototypes are never intended for business use and are generally crippled in one way or another to prevent them from being mistakenly used as production components by end-users.

ALLOWED VARIATIONS

In some cases, additional information is made available to the development team that requires changes in the outputs of previous stages. In this case, the development effort is usually suspended until the changes can be reconciled with the current design and the new results are passed down the chain until the project reaches the point where it was suspended.

The PER and PDR may, at their discretion, allow the development effort to continue while previous stage deliverables are updated in cases where the impacts are minimal and strictly limited in scope. In this case, the changes must be carefully tracked to make sure all impacts are appropriately handled.

REFERENCES

The following standards were used as guides to develop this SDLC description. The standards were reviewed and a tailored approach was developed to fit the specific requirements of database projects.

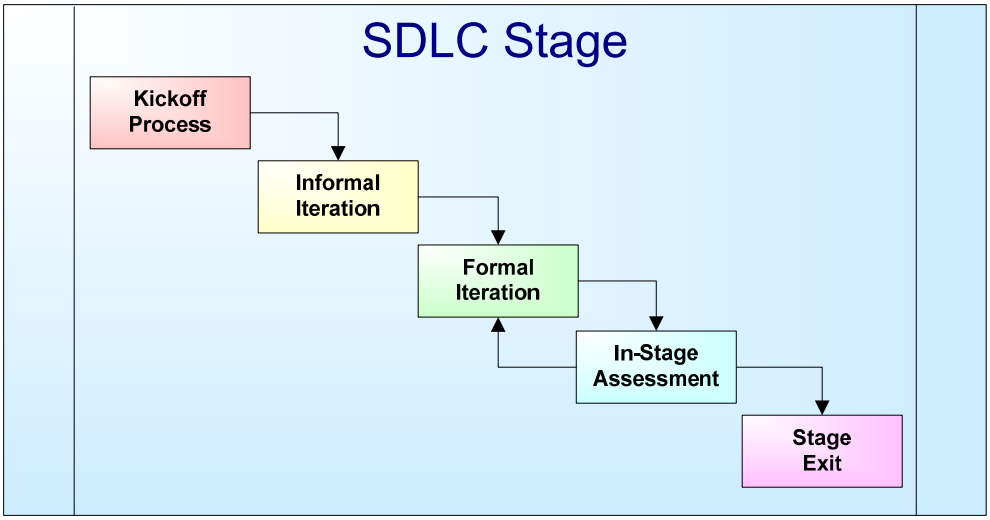
- ANSI/IEEE 1028: Standard for Software Reviews and Audits.
- ANSI/IEEE 1058.1: Standard for Software Project Management Plans.
- ANSI/IEEE 1074: Standard for Software Lifecycle Processes.
- SEI/CMMI: DAR, IT, OPD, IPM, RD, REQM, IPM, PI, PMC, PP, PPQA, TS, VER, and VAL process areas.

This document makes extensive use of terminology that is specific to software engineering. A glossary of standard software engineering terms is available online at:

www.shellmethod.com/refs/seglossary.pdf

GENERIC STAGE

Each of the stages of the development lifecycle follows five standard internal processes. These processes establish a pattern of communication and documentation intended to familiarize all participants with the current situation and thus minimize risk to the current project plan or component iteration plan. This generic stage description is provided to avoid repetitive descriptions of these internal processes in each of the following software lifecycle stage descriptions. The five standard processes are Kickoff, Informal Iteration, Formal Iteration, In-Stage Assessment, and Stage Exit.



KICKOFF PROCESS

Each stage is initiated by a kickoff meeting, which can be conducted either in person or by Web teleconference. The purpose of the kickoff meeting is to review the output of the previous stage, go over any additional inputs required by that particular stage, examine the anticipated activities and required outputs of the current stage, review the current project schedule, and review any open issues. The PDR is responsible for preparing the agenda and materials to be presented at this meeting. All project participants are invited to attend the kickoff meeting for each stage.

INFORMAL ITERATION PROCESS

Most of the creative work for a stage occurs here. Participants work together to gather additional information and refine stage inputs into draft deliverables. Activities of this stage may include interviews, meetings, the generation of prototypes, and electronic correspondence. All of these communications are deemed informal and are not recorded as minutes, documents of record, controlled software, or official memoranda.

The intent here is to encourage, rather than inhibit the communication process. This process concludes when the majority of participants agree that the work is substantially complete and it is time to develop the appropriate documentation.

FORMAL ITERATION PROCESS

In this process, draft deliverables are generated for formal review and comment. Each deliverable was introduced during the kickoff process and is intended to satisfy one or more outputs for the current stage. Each draft deliverable is given a version number and placed under configuration management control.

As participants review the draft deliverables, they are responsible for reporting errors found and concerns they may have to the PDR via electronic mail or, if available, a project-specific collaboration site. The PDR in turn consolidates these reports into a series of issues associated with a specific version of a deliverable. The person in charge of developing the deliverable works to resolve these issues and then releases another version of the deliverable for review. This process iterates until all issues are resolved for each deliverable. There are no formal check-off / signature forms for this part of the process. The intent here is to encourage review and feedback.

At the discretion of the PDR and PER, certain issues may be reserved for resolution in later stages of the development lifecycle. These issues are disassociated from the specific deliverable and tagged as "open issues." Open issues are reviewed during the kickoff meeting for each subsequent stage.

Once all issues against a deliverable have been resolved or moved to open status, the final (release) draft of the deliverable is prepared and submitted to the PDR. When final drafts of all required stage outputs have been received, the PDR reviews the final suite of deliverables, reviews the amount of labor expended against this stage of the project and uses this information to update the project plan.

The project plan update includes a detailed list of tasks, their duration and estimated level of effort for the next stage. The stages following the next stage

(out stages) in the project plan are updated to include a high level estimate of schedule and level of effort, based on current project experience.

Out stages are maintained at a high level in the project plan and are included primarily for informational purposes; experience has shown that it is very difficult to accurately plan detailed tasks and activities for the out stages in a software development effort. The updated project plan and schedule is a standard deliverable for each stage of the project. The PDR then circulates the updated project plan and schedule for review and comment and iterates these documents until all issues have been resolved or moved to open status.

Once the project plan and schedule has been finalized, all final deliverables for the current stage are made available to all project participants and the PDR initiates the next process.

IN-STAGE ASSESSMENT PROCESS

This is the formal quality assurance review process for each stage. This process is initiated when the PDR schedules an in-stage assessment with a selected End-user Reviewer (usually a Subject Matter Expert), a selected Technical Reviewer, and an independent Quality Assurance Reviewer (QAR).

These reviewers formally review each deliverable to make judgments as to the quality and validity of the work product, as well as its compliance with the standards defined for deliverables of that class. Deliverable class standards are defined in the Software Quality Assurance Plan (SQAP).

The End-user Reviewer is tasked with verifying the completeness and accuracy of the deliverable in terms of desired software functionality. The Technical Reviewer determines whether the deliverable contains complete and accurate technical information.

The QA Reviewer is tasked solely with verifying the reviews performed by the End-User and Technical reviewers. The QAR determines whether all reviews indicate substantial or unconditional occurrence and whether any conflicts exist in the review comments.

Each reviewer follows a formal checklist during their review, indicating their level of concurrence with each review item in the checklist. Refer to the SQAP for this project for deliverable class standards and associated review checklists. A deliverable is considered to be acceptable when each reviewer indicates substantial or unconditional concurrence with the content of the deliverable and the review checklist items.

New issues that represent significant changes to previously confirmed goals, requirements, or design elements are “binned” to a separate issues list for

consideration by the Configuration Control Board (CCB) at the planning session for the next version of the component or project. Issues submitted by reviewers against a specific deliverable that represent in-scope clarifications and corrections are relayed to the personnel responsible for generation of the deliverable. The revised deliverable is then re-released to project participants for review. Once all issues for the deliverable have been addressed, the deliverable is resubmitted to the reviewers for reassessment. Once all three reviewers have indicated concurrence with the deliverable, the PDR releases a final in-stage assessment report and initiates the next process.

STAGE EXIT PROCESS

The stage exit is the vehicle for securing the concurrence of principal project participants to continue with the project and move forward into the next stage of development. The purpose of a stage exit is to allow all personnel involved with the project to review the current project plan and stage deliverables, provide a forum to raise issues and concerns, and to ensure an acceptable action plan exists for all open issues.

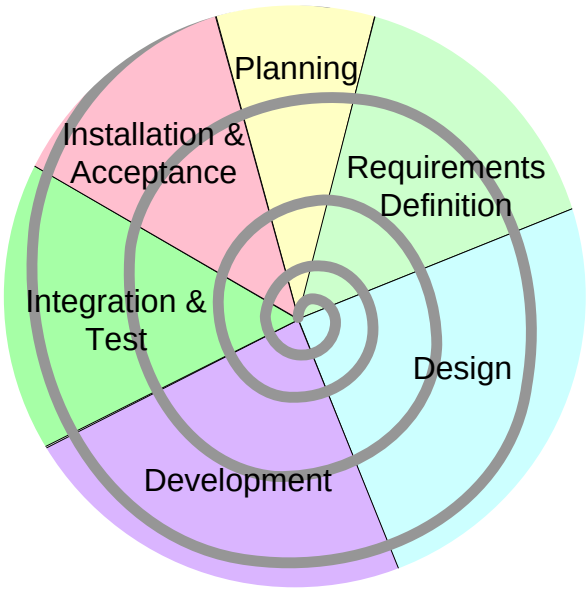
The process begins when the PDR notifies all project participants that all deliverables for the current stage have been finalized and approved via the In-stage Assessment Report (IAR). The PDR then schedules a stage exit review with the project executive sponsor and the PER as a minimum. All interested participants are free to attend the review as well. This meeting may be conducted in person or via Web teleconference.

The stage exit process ends with the receipt of concurrence from the designated approvers to proceed to the next stage. This is generally accomplished by receiving e-mail confirmation from project executive sponsor.

SDLC STAGES

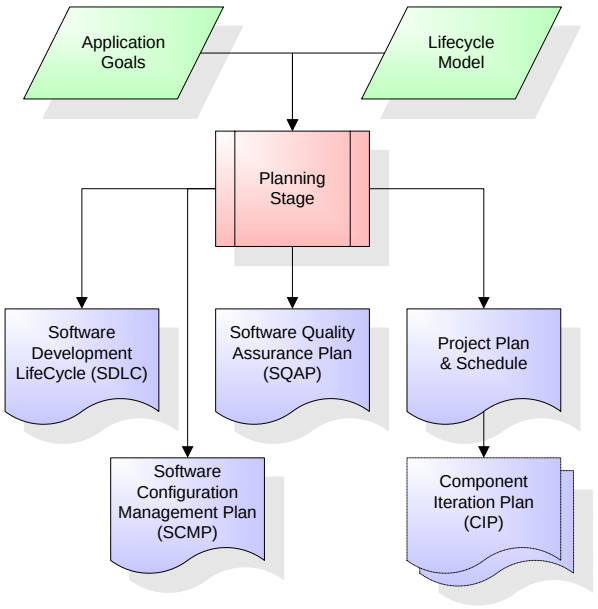
OVERVIEW

The six stages of the iterative lifecycle take a project or component through a complete development cycle, from initial planning through to production. Each iteration is tightly scope limited, with a goal of completing a full iteration in six to eight weeks.



PLANNING STAGE

The planning stage establishes a bird's eye view of the intended software product and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches.



The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included.

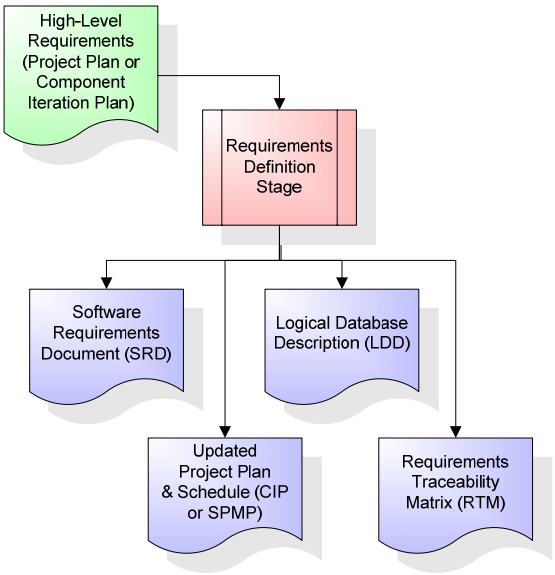
The outputs of the project planning stage are this SDLC description, the Software Configuration Management Plan (SCMP), the Software Quality Assurance Plan (SQAP), the Software Project Management Plan (SPMP) and the associated project or component schedule. The schedule includes a detailed listing of activities for the upcoming Requirements stage and high-level estimates of effort for the out stages. In the case of a component-based project, two or more Component Iteration Plans (CIPs) will flow down a subset of the goals in the Project Plan for subsequent refinement into component-specific documentation chains. Each CIP is limited to goals and constraints specific to a component. At this point, the project splits into multiple lifecycles, each dedicated to a specific component. These component lifecycles may be executed sequentially or in parallel, depending on resource availability.

REQUIREMENTS DEFINITION STAGE

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan or component iteration plan, as applicable. Each goal is refined into a set of requirements.

These requirements define the major functions of the intended application (generally termed operational data areas and reference data areas) and define the initial data entities. Major functions include critical processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities.

Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.



These requirements are fully described in the primary deliverables for this stage: the Software Requirements Document (SRD) and the Logical Database Description (LDD). The SRD contains complete descriptions of each requirement, including references to external documents, such as Use Cases. The LDD describes the major data entities of the project or component, along with their relationships to other entities and their user base. Note that detailed listings of database tables and fields are *not* included in the SRD or LDD.

The identifier associated with each requirement is also placed into the first version of the Requirements Traceability Matrix (RTM), along with the identifier of each goal from the parent project plan or component iteration plan. The purpose of the

RTM is to show that the artifacts developed during each stage of the software development lifecycle are fully linked to the artifacts developed in prior stages.

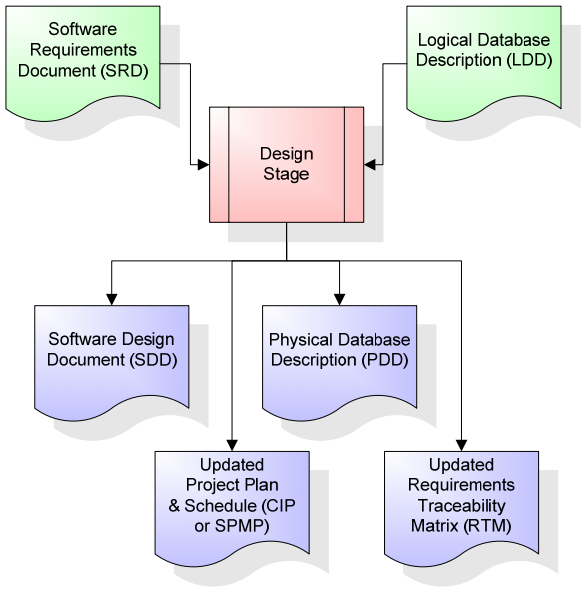
In the requirements stage, the RTM consists of a row of high-level requirements or goals, by goal identifiers, with a column of associated requirements for each goal, listed by requirement identifier. In the intersections of this matrix, the RTM shows that each requirement developed during this stage is formally linked to a specific project or component goal. In this format, each requirement can be traced to a specific goal, hence the term *requirements traceability*.

The outputs of the requirements definition stage include the SRD, the LDD, the RTM, and an updated CIP or SPMP.

DESIGN STAGE

The design stage takes as its initial input the requirements identified in the approved SRD and LDD. For each requirement, a set of one or more design elements are produced as a result of interviews, workshops, and/or prototype efforts.

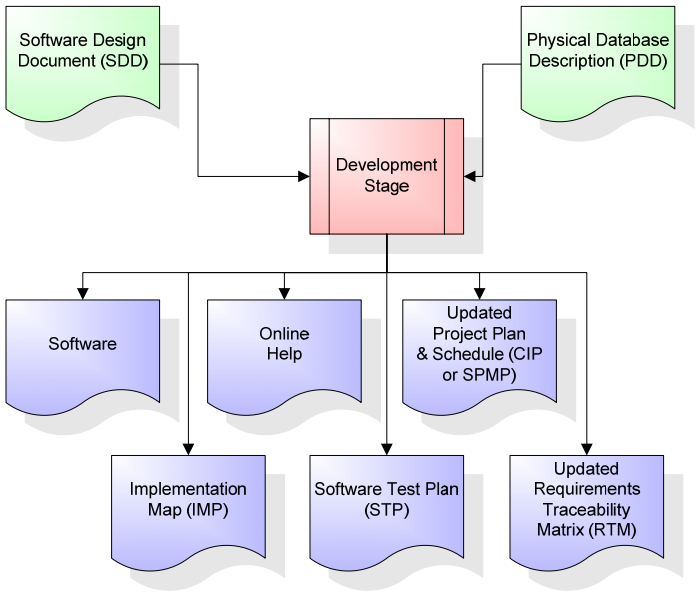
Design elements describe the desired software features in detail. The Software Design Document (SDD) contains the functional or dynamic design elements, such as functional hierarchy diagrams, screen layout diagrams, business rules, business process diagrams, and pseudocode. The Physical Database Description (PDD) contains the static or structural design elements such as the entity-relationship diagram, the access control matrix, and the data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.



When the SDD and PDD are finalized and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement. The outputs of the design stage are the SDD, the PDD, an updated RTM, and an updated CIP or SPMP.

DEVELOPMENT STAGE

The development stage takes as its inputs the design elements described in the approved SDD and PDD. For each design element, a set of one or more software artifacts are produced. Software artifacts include but are not limited to menus, dialogs, data entry forms, data reporting formats, and specialized procedures and functions. Appropriate test cases are developed for each set of functionally related software artifacts and an online help system is developed to guide users in their interactions with the software.



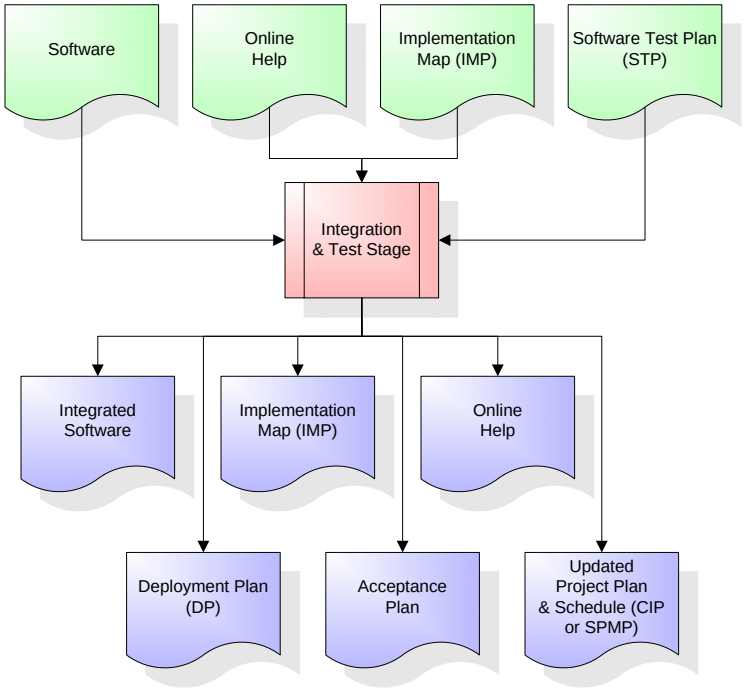
The RTM is updated to show that each developed artifact is linked to a specific design element and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration.

The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software, an Implementation Map (IMP) that identifies the primary code entry points for all major system functions, a Software Test Plan (STP) that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated CIP or SPMP.

INTEGRATION & TEST STAGE

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability.

During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user list are compiled into the Deployment Plan (DP).

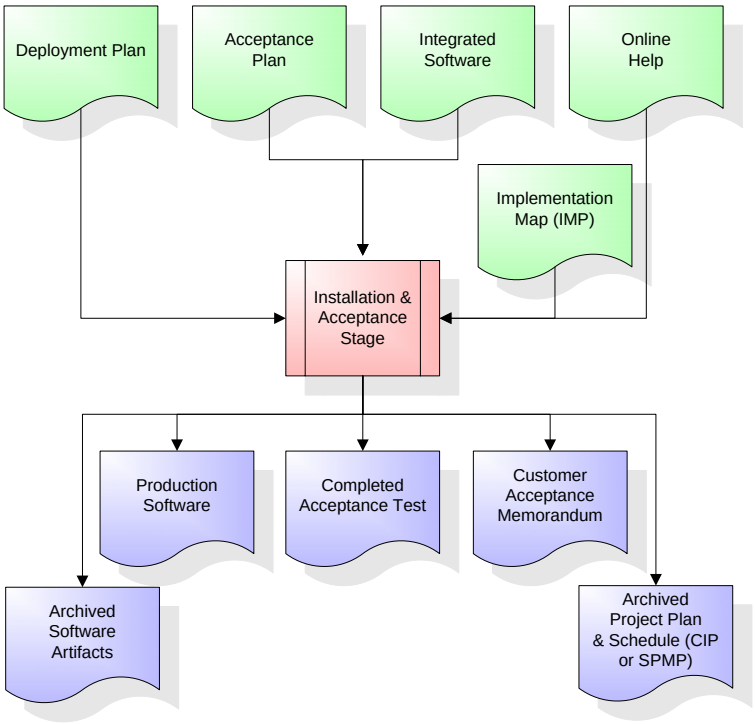


The outputs of the integration and test stage include an integrated set of software, an online help system, an updated IMP (if necessary), a DP that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated CIP or SPMP.

INSTALLATION & ACCEPTANCE STAGE

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loaded onto the production server as specified by the deployment plan. At this point, acceptance test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer.

After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.



The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR enters the last of the actual labor data into the component or project schedule and locks the component or project as a permanent project record by archiving all software items, the implementation map, the source code files, and the documentation for future reference.

KEY CONCEPTS

The structure imposed by this SDLC is specifically designed to maximize the probability of a successful software development effort. To accomplish this, the SDLC relies on five primary concepts:

- Domain Segregation
- Scope Restriction
- Progressive Enhancement
- Pre-defined Structure
- Incremental Planning

These concepts combine to mitigate the most common risks associated with software development efforts.

DOMAIN SEGREGATION

Larger projects are broken into a set of components, with each component focusing on a specific business domain. The end-user community for each component is composed of domain-specific subject matter experts. Participants for a specific component are specialists in the domain covered by the component, and naturally maintain a higher quality of interaction and level of interest in the development effort.

SCOPE RESTRICTION

The project scope is established by the contents of high-level requirements, also known as goals, incorporated into the SPMP and optional CIPs. These goals are subsequently refined into requirements, then design elements, then software artifacts.

This hierarchy of goals, requirements, design elements, and artifacts is documented in a Requirements Traceability Matrix (RTM). The RTM serves as a control element to restrict the project to the originally defined scope.

Project participants are restricted to addressing those requirements, elements, and artifacts that are directly traceable to product goals. New ideas and significant changes to the high-level requirements are "binned" to a separate issues list for consideration by the Configuration Control Board (CCB) at the planning session for the next version of the component or project. This minimizes scope creep, which is a leading cause of software project failure.

PROGRESSIVE ENHANCEMENT

Each stage of the SDLC takes the outputs of the previous stage as its initial inputs. Additional information is then gathered, using methods specific to each stage. As a result, the outputs of the previous stage are progressively enhanced with additional information.

By establishing a pattern of enhancing prior work, the project precludes the insertion of additional requirements in later stages. New requirements are formally set aside by the development team for later reference, rather than going through the effort of backing the new requirements into prior stage outputs and reconciling the impacts of the additions. As a result, the project participants maintain a tighter focus on the original product goals, minimize the potential for scope creep, and show a preference for deferring out-of-scope enhancements, rather than attempting to incorporate them into the current effort.

PRE-DEFINED STRUCTURE

Each stage has a pre-defined set of standard processes, such as Informal Iteration and In-Stage Assessment. The project participants quickly grow accustomed to this repetitive pattern of effort as they progress from stage to stage. In essence, these processes establish a common rhythm, or culture, for the project.

This pre-defined structure for each stage allows the project participants to work in a familiar environment, where they know what happened in the past, what is happening in the present, and have accurate expectations for what is coming in the near future. This engenders a high comfort level, which in turn generates a higher level of cooperation between participants. Participants tend to provide needed information or feedback in a timelier manner, and with fewer miscommunications. This timely response pattern and level of communications quality becomes fairly predictable, enhancing the ability of the PDR to forecast the level of effort for future stages.

INCREMENTAL PLANNING

The entire intent of incremental planning is to minimize surprises, increase accuracy, provide notification of significant deviations from plan as early in the

SDLC as possible, and coordinate project forecasts with the most current available information.

In this SDLC, the project planning effort is restricted to gathering metrics on the current stage, planning the next stage in detail, and restricting the planning of later stages, also known as Out Stages, to a very high level. The SPMP or CIP is updated as each stage is completed; current costs and schedule to date are combined with refined estimates for activities and level of effort for the next stage.

The activities and tasks of the next stage are defined only after the deliverables for the current stage are complete and the current metrics are available. This allows the planner to produce a highly accurate plan for the next stage. Direct experience has shown that it is very difficult to develop more than a cursory estimate of anticipated structure and level of effort for out stages.

The estimates for out stages are included to allow a rough estimate of ultimate project cost and schedule. The estimates for out stages are reviewed and revised as each stage is exited. As a result, the total project estimate becomes more and more accurate over time.

As each stage is exited, the updated project plan and schedule is presented to the customer. The customer is apprised of project progress, cost and schedule, and the actual metrics are compared against the estimates. This gives the customer the opportunity to confirm the project is on track, or take corrective action as necessary. The customer is never left “in the dark” about the progress of the project.