The Software Development Life Cycle (SDLC)

For Small To Medium Database Applications

Document ID: REF-0-02
Version: 1.0d

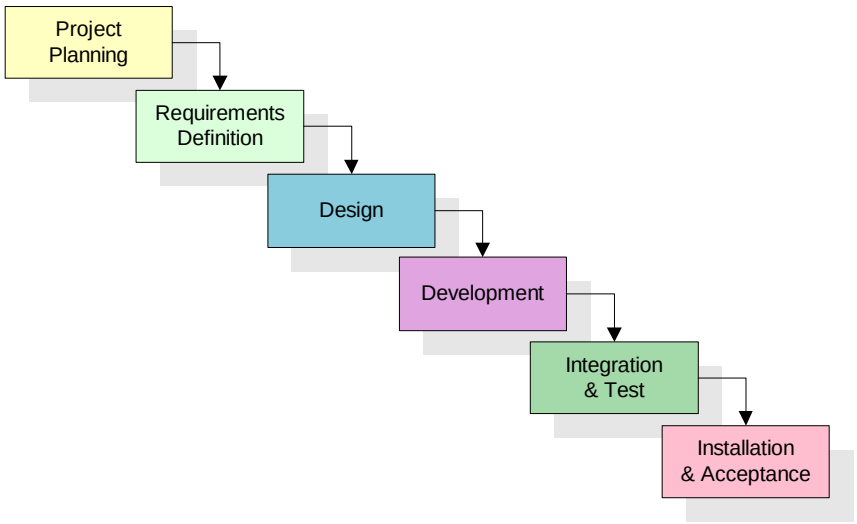
TABLE OF CONTENTS

INTRODUCTION

This document describes the Software Development LifeCycle (SDLC) for small to medium database application development efforts. This chapter presents an overview of the SDLC, alternate lifecycle models, 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.

THE SDLC WATERFALL

Small to medium database software projects are generally broken down into six stages:



The relationship of each stage to the others can be roughly described as a waterfall, where the outputs from a specific stage serve as the initial inputs for the following stage.

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.

After the project is completed, the Primary Developer Representative (PDR) and Primary End-User Representative (PER), in concert with other customer and development team personnel develop a list of recommendations for enhancement of the current software.

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 module then it is to develop the module 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 modules 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 waterfall 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 their impacts are appropriately handled.

OTHER SDLC MODELS

The waterfall model is one of the three most commonly cited lifecycle models. Others include the Spiral model and the Rapid Application Development (RAD) model, often referred to as the Prototyping model.

SPIRAL LIFECYCLE

The spiral model starts with an initial pass through a standard waterfall lifecycle, using a subset of the total requirements to develop a robust prototype. After an evaluation period, the cycle is initiated again, adding new functionality and releasing the next prototype. This process continues, with the prototype becoming larger and larger with each iteration. Hence, the "spiral."

The theory is that the set of requirements is hierarchical in nature, with additional functionality building on the first efforts. This is a sound practice for systems where the entire problem is well defined from the start, such as modeling and simulating software. Business-oriented database projects do not enjoy this advantage. Most of the functions in a database solution are essentially independent of one another, although they may make use of common data. As a result, the prototype suffers from the same flaws as the prototyping lifecycle described below. For this reason, the software development team has decided against the use of the spiral lifecycle for database projects.

RAPID APPLICATION DEVELOPMENT (RAD) / PROTOTYPING LIFECYCLE

RAD is, in essence, the "try before you buy" approach to software development. The theory is that end users can produce better feedback when examining a live system, as opposed to working strictly with documentation. RAD-based development cycles have resulted in a lower level of rejection when the application is placed into production, but this success most often comes at the expense of a dramatic overruns in project costs and schedule.

The RAD approach was made possible with significant advances in software development environments to allow rapid generation and change of screens and other user interface features. The end user is allowed to work with the screens online, as if in a production environment. This leaves little to the imagination, and a significant number of errors are caught using this process.

The down side to RAD is the propensity of the end user to force scope creep into the development effort. Since it seems so easy for the developer to produce the basic screen, it must be just as easy to add a widget or two. In most RAD lifecycle failures, the end users and developers were caught in an unending cycle of enhancements, with the users asking for more and more and the developers trying to satisfy them. The participants lost sight of the goal of producing a basic, useful system in favor of the siren song of glittering perfection.

For this reason, the software development team does not use a pure RAD approach, but instead blends limited prototyping in with requirements and design development during a conventional waterfall lifecycle. The prototypes developed are specifically focused on a subset of the application, and do not provide an integrated interface. The prototypes are used to validate requirements and design elements, and the development of additional requirements or the addition of user interface options not readily supported by the development environment is actively discouraged.

REFERENCES

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

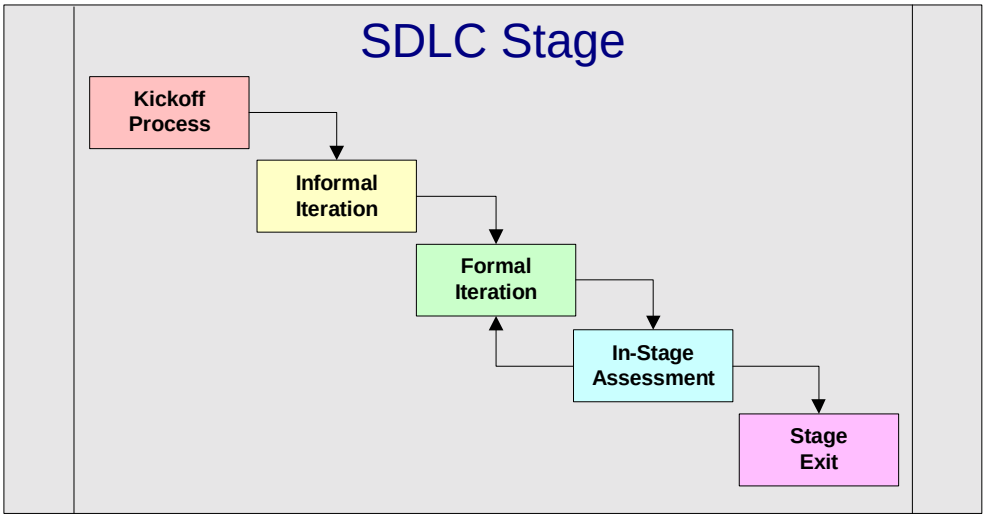
- ANSI/IEEE 1028: Standard for Software Reviews and Audits
- ANSI/IEEE 1058.1: Standard for Software Project Management Plans
- ANSI/IEEE 1074: Standard for Software Lifecycle Processes
- SEI/CMM: Software Project Planning Key Process Area

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

- http://www.elucidata.org/refs/seglossary.pdf

GENERIC STAGE

Each of the stages of the development lifecycle follow 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. 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 generate draft deliverables for formal review and comment.

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. 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, 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 schedule 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; direct experience has shown that it is very difficult to accurately plan detailed tasks and activities for out stages in a software development lifecycle. 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. In a small software development project, the deliverables for each stage are generally small enough that it is not cost effective to review them for compliance with quality assurance standards before the deliverables have been fully developed. As a result, only one in-stage assessment is scheduled for each stage.

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

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 section of the project plan.

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 completeness and compliance of the deliverable against the associated deliverable class standard. The QAR may make recommendations, but cannot raise formal issues that do not relate to the deliverable standard.

Each reviewer follows a formal checklist during their review, indicating their level of concurrence with each review item in the checklist. Refer to the software quality assurance plan 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.

Any issues raised by the reviewers against a specific deliverable will be logged and relayed to the personnel responsible for generation of the deliverable. The revised deliverable will then be released to project participants for another formal review iteration. Once all issues for the deliverable have been addressed, the deliverable will be resubmitted to the reviewers for reassessment. Once all three reviewers have indicated concurrence with the deliverable, the PDR will release a final in-stage assessment report and initiate 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.

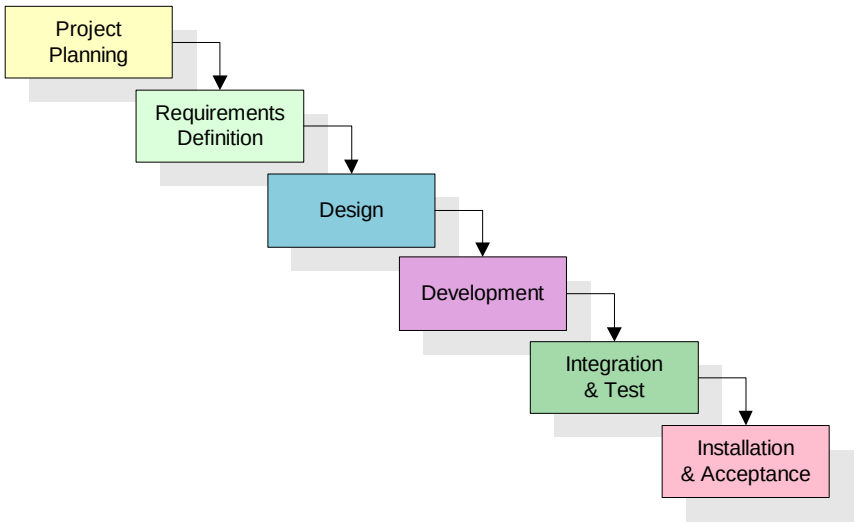
The process begins when the PDR notifies all project participants that all deliverables for the current stage have been finalized and approved via the In-Stage Assessment report. The PDR then schedules a stage exit review with the project executive sponsor and the PER as a minimum. All interested participants 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 entering the minutes of the exit review as a formal document of record, with either physical or digital signatures of the project executive sponsor, the PER, and the PDR.

SDLC STAGES

OVERVIEW

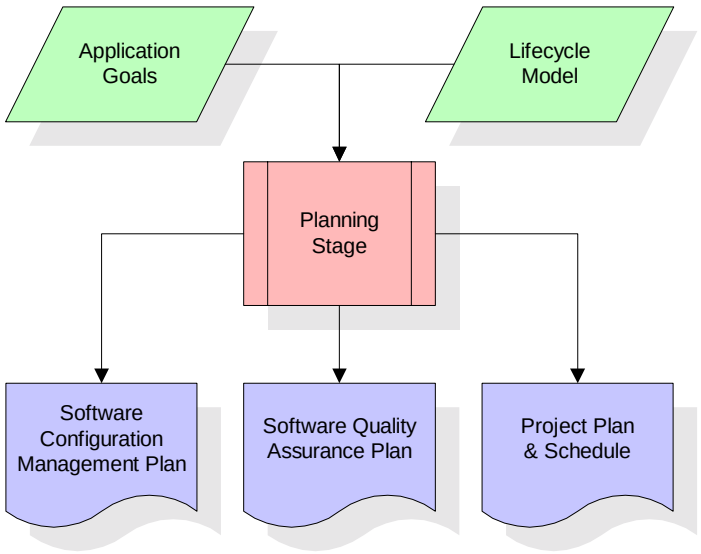
The six stages of the SDLC are designed to build on one another, taking the outputs from the previous stage, adding additional effort, and producing results that leverage the previous effort and are directly traceable to the previous stages. This top-down approach is intended to result in a quality product that satisfies the original intentions of the customer.



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 Waterfall SDLC.

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.

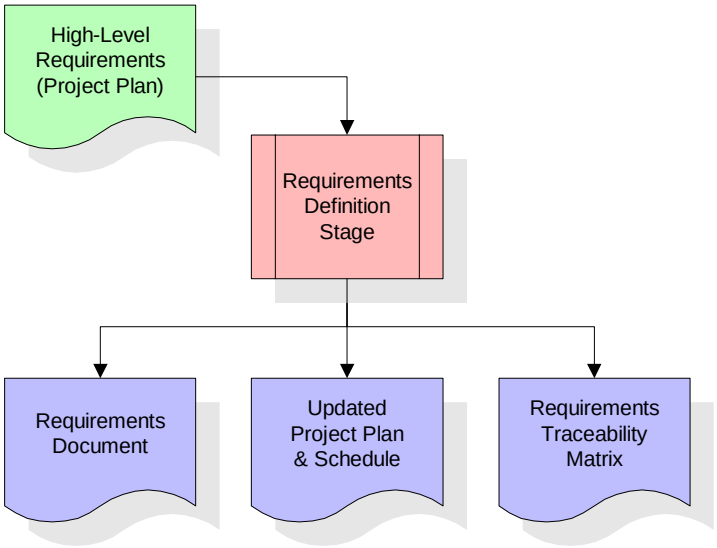
The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage, and high-level estimates of effort for the out stages.

REQUIREMENTS DEFINITION STAGE

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be refined into a set of one or more requirements.

These requirements define the major functions of the intended application, define 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 Requirements Document and the Requirements Traceability Matrix (RTM). the requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are *not* included in the requirements document.

The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

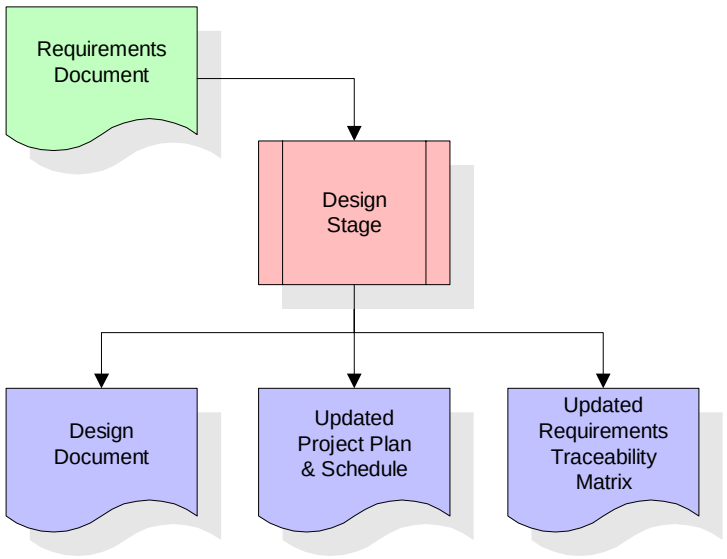
In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term *requirements traceability*.

The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

DESIGN STAGE

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

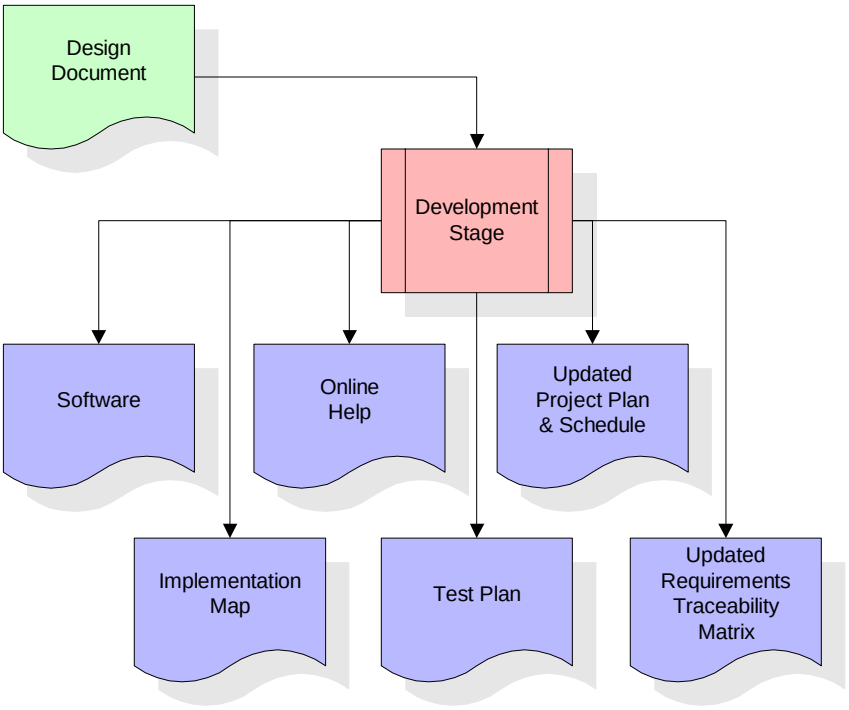
Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudocode, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.



When the design document is 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 design document, an updated RTM, and an updated project plan.

DEVELOPMENT STAGE

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



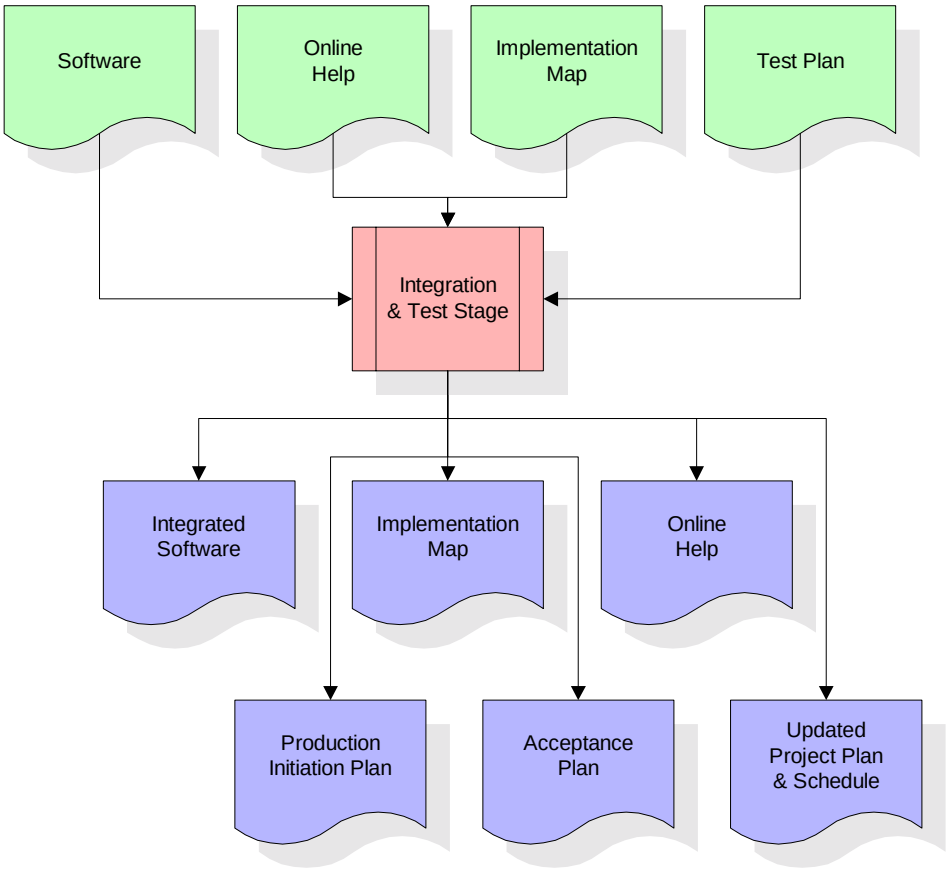
The RTM will be updated to show that each developed artifact is linked to a specific design element, and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration.

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 that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

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 Production Initiation Plan.

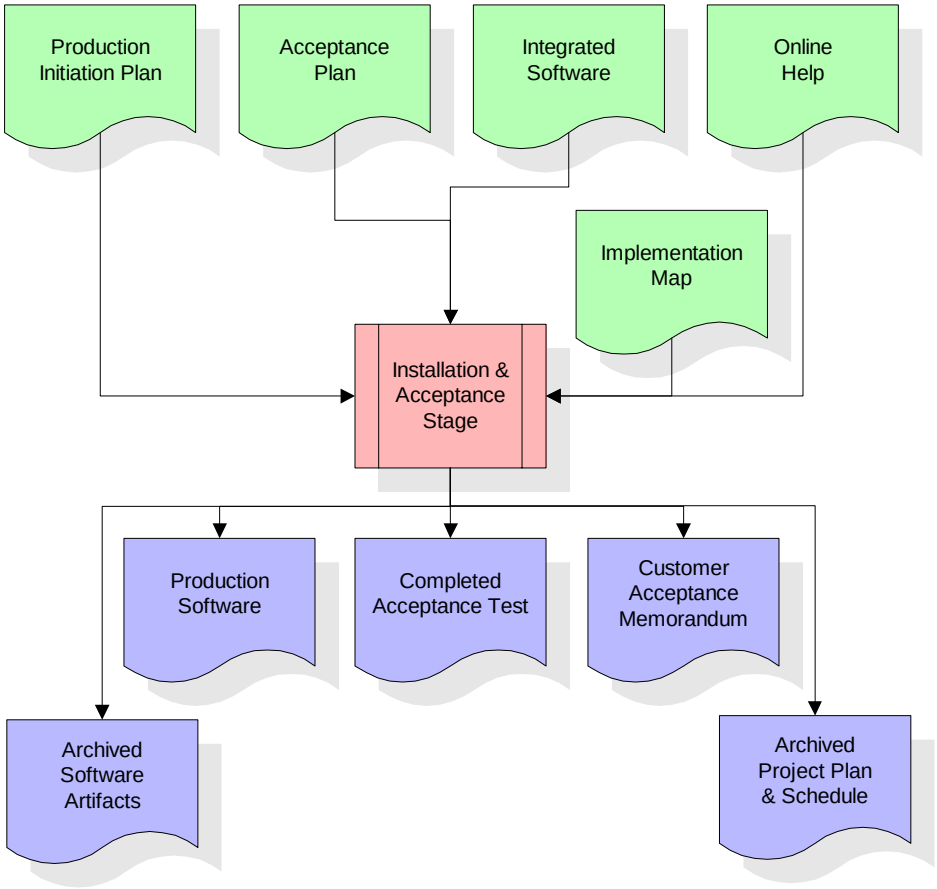


The outputs of the integration and test stage include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated project plan.

INSTALLATION & ACCEPTANCE STAGE

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loaded onto the production server. At this point, all 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 project schedule and locks the project as a permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.

CONCLUSION

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 four primary concepts:

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

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

SCOPE RESTRICTION

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

This hierarchy of goals, requirements, 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. This prevents the substantial occurrence of scope creep, which is the 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 more timely 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 project plan 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.