| [Print](javascript:window.print()) | [Close](javascript:parent.window.close()) |

**Course Transcript**

The Waterfall Software Development Model

**The Application Development Lifecycle**

| [1. Course Introduction](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t2) |

| [2. The Application Development Lifecycle (ADLC)](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t6) |

| [3. ADLC Startup Phases](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t10) |

| [4. ADLC Product Phases](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t14) |

**Software Development Lifecycle**

| [1. The Software Development Lifecycle (SDLC)](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t19) |

| [2. SDLC Umbrella Tasks](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t23) |

| [3. SDLC Analysis Phase](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t27) |

| [4. SDLC Design Phase](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t31) |

| [5. SDLC Implementation Phase](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t35) |

| [6. SDLC Validation Phase](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t39) |

| [7. SDLC Deployment Phase](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t43) |

| [8. SDLC Cost of Change](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t47) |

**Sequential, Iterative and Agile Models**

| [1. The Sequential Approach Software Method](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t52) |

| [2. Sequential Approach Software Models](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t56) |

| [3. The Iterative Approach Software Method](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t60) |

| [4. Agile Software Models](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#t64) |

Course Introduction

Learning Objective

*After completing this topic, you should be able to*

* *start the course*

**1. Introduction to the course**

Waterfall is a traditional sequential software development model. In this course, Brigitte Birze will examine the ADLC approach, as well as the SDLC approach, and she'll console contrasts traditional Waterfall development with iterative approaches, such as Agile. I hope you enjoy the course.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

The Application Development Lifecycle (ADLC)

Learning Objective

*After completing this topic, you should be able to*

* *describe the Application Development Lifecycle (ADLC) in Software Practices (Waterfall)*

**1. Phases of ADLC**

The Application Development Lifecycle, sometimes referred to as the system lifecycle, is a framework that describes the entire lifecycle of a product or a system, from inception through planning, implementation, production, and end of life. The system lifecycle framework was created to optimize product development, reducing not only cost, but time-to-market as well. The Application Development Lifecycle is broken down into distinct phases, each with their own well-defined functions and activities. The phases follow each other in order, with the work produced by earlier phases feeding into later phases as their input. The first phase is Initiation, where a need or opportunity is identified and someone comes up with an idea for a solution. The idea is used to create a concept proposal to pitch to the stakeholders. If the stakeholders like the idea, it moves to the next phase, system concept development. During system concept development, the concept is fleshed out and bounded by defining the scope. The concept is researched to produce several different studies. The stakeholders will review these studies to determine if a project should be kicked off to create the concept. Some studies produced during this phase include a cost-benefit analysis to determine return on investment, the risk management plan, and feasibility studies.   
*Heading: The Application Development Lifecycle (ADLC).  
  
The ADLC involves the entire lifecycle of product or system. It is broken into distinct phases each with clearly defined activities and each phase's outputs feed into later phases.  
  
The lifecycle starts with the Initiation phase followed by System Concept, Planning, Analysis, Design, Development, Validation, Deployment, Production, and lastly End of Life phase.*   
  
Once the stakeholders kick off the project, the Planning phase activities begin. The project management plan must be developed to guide the project execution and control. The plan will outline the resources needed to create the concept, such as hardware, software, skilled workers, and logistics for the work environment. The results of the planning will be used to refine the cost projections, identify milestones, and create the project timeline. Once the stakeholders approve the plan, they release the money to fund the project. The next phase is Analysis, where the system concept is used to create the architectural and functional requirements, stating exactly what must be built and how it will function. The requirements drive the development of the software and are the basis for the customer acceptance testing at the end of the project. Once the requirements are in place, stating what must be built, the developers design the application, describing how it will be implemented to satisfy the requirements. Detailed design specification includes software and database design, as well as user interface mock-ups. The quality assurance team also uses the requirements to create the test plan, describing how the in-system will be validated.   
  
The detailed design is used in the Development phase to finally implement the application. The team uses several different development environments. Each will contain all the required hardware and software support tools. The system code and database are created, along with the unit test, and system integration is done. The quality assurance team also uses the detailed design to write test cases they will use to exercise the engineering code during validation. During the Validation phase, the engineering software is handed off to the quality assurance test engineers to be verified. The software is exercised against the test cases, to ensure it is robust and satisfies the requirements. Once the software is deemed ready, it enters the Deployment phase. In this phase, the software is deployed into the production environment. This could be a stepped phase, involving small groups and alpha and beta testing, to allow resolution of issues before full production. Deployment involves training and all the logistics that go into putting a product or system into production. While in production, the application will need operational support and maintenance, as issues are found or new features are desired.   
  
The End of Life phase involves taking the application out of production and determining how to dispose of any sensitive data. The data must be destroyed, archived, or transitioned to a new system. The Analysis, Design, Development, Validation, and Deployment phases, taken together, are known as the Software Development Lifecycle, or SDLC. The SDLC phases are where the system is taken from an idea to production software, which realizes the product requirements. The Application Development Lifecycle includes one or more SDLC iterations. The first SDLC iteration is where the application is built from the system concept – the initial development.Once the application goes into production, there are often multiple modifications that must be made to maintain the software. The application should be consistently evaluated to ensure it is meeting the needs of the customer. New projects can be initiated to handle upgrades, fix issues, or add new features as needs are uncovered. Each development project initiated during production will have its own Software Development Lifecycle. The team will determine what it needs to create and then design and produce software that is validated. The validated software will then be deployed back into production.   
*Heading: ADLC and SDLC.  
  
There are multiple SDLC phases in the ADLC.  
  
The first is initial development.  
  
There are several SDLC phases during operating: new features, upgrades, and bug fixes.  
  
In the illustration, initiation feeds into system concept. System concept feeds into planning, which leads to the SDLC. This leads to production and the SDLC, and finally to the End of Life phase.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

ADLC Startup Phases

Learning Objective

*After completing this topic, you should be able to*

* *use Application Development Lifecycle Startup phases in Software Practices (Waterfall)*

**1. The ADLC Startup phases**

The Application Startup phases include Initiation, System Concept, and Planning. All Startup phase activities involve investigation and analysis. The inception of an application lifecycle is caused by an idea, a problem or an issue, which starts the Initiation phase. This phase is driven by a single sponsor, normally a product manager, who investigates a customer or business need or an innovative idea. Analysis uncovers several approaches to satisfy the need, or produce the innovation. The Initiation phase answers the question, "Why do the project?" The System Concept Development phase puts boundaries around the concept, scoping it. Preliminary analysis is done to flush out the concept and prove its viability. Preliminary analysis will determine the company's objectives, and the cost and benefits of going forward. Alternative solutions may be proposed with recommendations. And the system concept will be presented to stakeholders for buy in. If approved, a project will be kicked off to implement the concept.   
*Heading: ADLC Start-Up Phases.  
  
All start-up phases require stakeholder buy in to continue.  
  
The start-up phase consists of the first three phases: Initiation, System Concept Development, and Planning.  
  
Initiation.  
  
The Initiation phase begins with an idea or the identification of a need or opportunity, and is driven by a sponsor.  
  
System Concept Development.  
  
This phase is used to define the scope of the project. Competitor surveys and reports, as well as feasibility studies with ROI are presented to stakeholders to facilitate buy-in.   
  
Planning.  
  
The Planning phase is where you determine what resources are needed, dates are set, and a cost projection is created.  
  
The diagram on the slide shows how these start-up phases feed into each other. The Planning phase feeds into SDLC, which feeds in to a cycle that consists of the Production phase involving SDLC, which feeds into the End of Life phase.*   
  
Before the project can be ramped up, intense planning must be done. The project manager is brought on to evaluate the resources needed to complete the project. The project will need development environments, which includes servers, workstations and networking, the compilers, editors and other software tools, as well as adequate power, cooling, and space to house the equipment. Multiple separate environments will be needed for individual developers, system integration, system testing, and final acceptance testing. And skilled workers will need to be brought onto the project to do the design, development and testing. Interaction with many company departments is required to support the project. These departments like Training, HR, Facilities, Technical Support and Operations, must be identified in the plan to facilitate coordination between departments. Planning also includes creation of the project timeline, taking into account marketing windows of opportunity like conferences and customer deadlines.   
  
The timeline will include milestones to mark the completion of significant units of work, such as the phases of the SDLC. The planning results are used to refine cost projection and determine a budget to run the project. Senior management must approve the plan, before the finance department can implement the budget to ramp up the project. The activities of each startup phase produce artifacts, and approve documents that are used as input to later phases in the project. The idea that started the Initiation phase, as well as the analysis done around it, are laid out in the concept proposal. This document tells the stakeholders why they should move forward with this project. System concept development involves a great deal of analysis to prove the concept is worthwhile. The results of the analysis are laid out in several support documents, including the feasibility study, cost benefit analysis, and the risk management plan. The feasibility study brings in research and analysis from many areas and sources, to describe the potential of implementing the concept.   
*Heading: System Start-Up Phase Artifacts.  
  
Phase activities produce artifacts that are input to later phases. SDLC phases can access all artifacts from System Investigation phases.  
  
The Initiation phase outputs the Concept Proposal document, which informs the System Concept phase.  
  
The System Concept phase outputs four artifacts: the statement of work, or SOW, the feasibility study, Cost Benefit Analysis, and Risk Management Plan. These inform the Planning phase.*   
  
Feasibility studies evaluate the concept from five different viewpoints, technical, economic, legal, operational, and scheduling. A very important area of analysis is the market and competitive surveys, which show a market exists for the concept and does an assessment of competing products. This section also includes how timing affects the concept's potential and recommendations. The cost benefit analysis compares the cost of implementing and deploying the concept, against the benefit the company will enjoy once it is deployed. The results indicate whether the return on investment is worthwhile, and by how much. The risk management plan describes the possible risk to the project, along with the impact and a plan to mitigate each one. The plan could be to take steps to avoid the risk, reducing the possibility of the risk occurring, to take no steps and accept the risk could occur, or to transfer the risk by outsourcing a portion of the project. The Statement of Work is the formal document that captures the purpose of the project, what the project will create and deliver, as well as the timeline, high-level requirements, budget, and acceptance criteria. The Statement of Work sets expectations and boundaries of the project, and is often used as a legal binding contract.   
  
Once the project is kicked off, the project manager begins creating the project management plan, which holds all the planning decisions and assumptions made during the Planning phase. These decisions include the project scope, cost, timeline with milestones, and development activities and the project resources required. The project management plan also lays out the project strategy and how it will be executed, managed, and controlled. This applies to all phases of the Software Development Lifecycle – requirements, design, development, validation, and deployment – as well as the SDLC Umbrella tasks, such as budget, quality, management, risk management, and interaction with other departments. The knowledge gained from intense planning is used to define the project's final budget, which is included in the plan. The project management plan is a formal document, which must be approved by the stakeholders, before the project can enter the SDLC phases and begin ramping up. All these artifacts created during the system Startup phases are used as inputs to the SDLC phases, where the product or system is implemented.   
*Heading: System Start-Up Phase Artifacts.  
  
The Planning phase produces the Project Management plan which informs the SDLC.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

ADLC Product Phases

Learning Objective

*After completing this topic, you should be able to*

* *use the Operation and Maintenance phase in Software Practices (Waterfall)*

**1. The Operation and Maintenance phase**

The Operation and Maintenance phase occurs after the initial development, SDLC, when the application is deployed to the customer environment. This is the longest and most expensive phase of the Application Development Lifecycle. While the initial SDLC to create an application may only last a year or two, the application can stay in the Operation and Maintenance phase for decades. Most of the cost of the application or system is incurred, while the system is in production, due to operation and maintenance activities. Operation is administration of the software; the cost of keeping the application running. In the case of an installed system, this includes the cost of the operators and administrators that perform backups and provide user assistance. It also includes the cost of upkeep of the environment, including the hardware servers, networks, software licenses, cooling, power, storage, and space for the equipment. For off-the-shelf software, the operational cost could include the helpdesk call center and the operators, and a website where customers can get information or download fixes, device drivers, and manuals.   
*Heading: Operations and Maintenance Phase.  
  
Operation and maintenance of software in production.  
  
Operation refers to the administration of the software and cost of operation of software that is in production and used by the customer.  
  
Maintenance involves the evaluation and assessment of the application. Maintenance of multiple SDLCs includes evaluation and assessment of adaptive changes, perfective changes, corrective changes, and preventative changes.  
  
Once the software is in production, it can cycle through the Operation and Maintenance phases many times before it moves to End of Life.*   
  
Applications in production should have a maintenance plan and a change control process, specifying how users should report issues or request updates. The process should also specify, how the helpdesk will accept, prioritize, document, and route requests. The operational application should be continually evaluated, to measure the effectiveness of the product. The application's performance must meet its service level agreement, which outlines the application's performance and reliability requirements. Maintenance also includes assessing the application, to ensure it has not become obsolete and is still working as expected, as well as performance analysis, to determine if changes are needed to accommodate current business processes. There are many changes that can occur in a customer environment, which require the application to adapt. This often leads to kicking off a new software project to implement the changes. While in production an application may require updates, to adapt to changes in the software environment, so it remains usable. For our installed system, this can include upgrades to the hardware or software, or new equipment to handle end of capacity situations with the system storage, network, or power needs.   
  
For an off-the-shelf product, this could be updates to accommodate a new version of the operating system, or to add Internet connectivity. An application may be updated to add enhancements, new features or implement new user requirements. This is often the case when customers begin to use the application, and form new ideas on how it could function more effectively, or when changes in the business processes, regulations or workflows require enhancements. Sometimes bugs or defects are found in operation, and the application must be modified to correct the faults discovered. Applications in production can also be updated as a preventative measure, to make software more reliable or prevent future issues. An example of preventative changes happened at the turn of the century. Most of the software in production was modified to ensure it would not be affected by the Y2K bugs. Evaluation of production software includes, determining when it is reaching the end of its usefulness and should be taken out of production. There are many reasons a company might kickoff a sunset initiative for an application. Within its Local Area Network, companies run many legacy applications, which provide little value, but require full operational and maintenance costs.   
*Heading: End of Life Phase.  
  
Application Sunset Initiative.  
  
A Sunset initiative is implemented against an application to consolidate redundant or obsolete systems, migrate an application to a less expensive platform, and save on operational costs.  
  
A Sunset Initiative requires a detailed plan on sensitive data decisions and protection, as well as how to transition to a new system.*   
  
Companies may want to sunset some of these applications to eliminate redundant or obsolete systems, or to consolidate several single function apps into a single large multifunction system, or to migrate from an expensive platform to a cheaper one. Sunsetting applications can save substantial money by eliminating the needs for ongoing maintenance. However, any sunset initiative needs a detailed plan for how to discard the system information, hardware and software as well as how to transition to the new system. Information contained in the system needs to be categorized to determine if it should be moved, archived, discarded, or destroyed. This process will identify data that should not be lost, data that is required for business reports, audits or for regulatory compliance needs to be archived or migrated to a new system. And the plan must explain how sensitive data will be protected. Sensitive data that is not needed must be destroyed. This includes backups and archives. The plan should also include recommendations for replacing the functionality of the software, that's reaching the End of Life. If a new system will be put in place, the transition must include retraining of the users and support workers on the new system.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

The Software Development Lifecycle (SDLC)

Learning Objective

*After completing this topic, you should be able to*

* *describe the Software Development Lifecycle (SDLC) framework in Software Practices (Waterfall)*

**1. The SDLC framework**

The Software Development Lifecycle is a framework, which describes the different phases in a software development project. Each phase in the framework has distinct activities and responsibilities, making the Software Development Lifecycle framework a model for the development and management of software projects. The Software Development Lifecycle is a framework for planning, creating, testing, and deploying a software application or system. The SDLC phases are project oriented. They are used as a model for the development and management of software projects. The SDLC begins after a project is initiated and planned, and ends once the software is deployed in production. Each phase in the SDLC has distinct functions and responsibilities. The work completed in earlier phases results in artifacts that are used as input to later phases. The Analysis phase includes in-depth research and analysis on the system concept and user needs. This knowledge is used to transform the high-level Statement of Work, to a detailed architectural and functional requirements document, stating exactly what must be accomplished.   
*Heading: The Software Development Lifecycle (SDLC) Phases.  
  
The Software Development Lifecycle (SDLC) is a framework describing software development phases.  
  
The phases contain specific tasks and activities and each phases's deliverables are used by later phases.  
  
Different software models implement tasks differently, specifically the Waterfall model and the Agile software processes.  
  
The SDLC starts with the Analysis phase, which feeds into the Design phase, into the Development phase, into the Validation phase, and finally on to the Deployment phase.*   
  
The Design phase uses the requirements document as input to create a detailed system design, stating exactly how the software will be built to satisfy the requirements. The designs created include the architectural, software, and database design, as well as user interface mockups. Other activities in this phase includes selection of software tools and quality assurance planning. All this work is detailed in the high-level and detailed design documents, and the quality assurance test plan. The Development phase is where the software engineers use a system design, to implement the software application. This work requires several development environments, each equipped with all the supporting software tools. Development activities in this phase include writing and testing code, code reviews, database creation and system integration. Quality assurance activities include writing test cases, creating test files and setting up an isolated test environment. The results of this phase is a working engineering software application, ready for validation and the QA test cases.   
  
During the Validation phase, the QA testers exercise the engineering software application, by running through their test cases. Test cases will validate the application against the requirements to ensure compliance, and identify any defects in the robustness or quality of the software. Validation is conducted by the test or QA team in an environment isolated from the development team. Defects identified are reported to development, which fixes the bugs and returns the updated software to test. When the software is deemed ready for production, the QA team records its finding in the test analysis report. The last phase of the SDLC, involves deploying the validated software application, into the production environment or through a supply chain to be sold as a product. This phase may involve additional internal and external acceptance testing, with small groups of customers to allow resolution of issues before full production. For IT systems, deployment may also include procurement of the application's production environment, including servers, software licenses, cooling, networking, power, and personnel to support the system.   
  
The transition to the new software includes operator, administrator, and user training and setting up the helpdesk support. The Software Development Lifecycle, defines all the tasks required for developing and deploying software. However, it does not specify how the phases and tasks must be implemented. There are many different software models and processes, which each implement the SDLC in their own unique way. In addition to the SDLC development or instream activities, a software project also contains management activities, commonly called umbrella tasks. Umbrella tasks are not contained within a single phase of the SDLC. They may span multiple phases and some are enforced throughout the entire life of the project. Project management activities includes scheduling, project tracking, and control.   
*Heading: SDLC Umbrella Tasks.  
  
The Umbrella tasks consist of management activities. The tasks are not contained within a single phase but rather span all SDLC phases.  
  
Umbrella tasks include training, CM, measurement, project management, tracking, documentation, QA, and risk management.*   
  
Project management is also responsible for planning, budget, and acquisition and retention of resources, both skilled workers and electronic and physical resources. Project management must also interface with other departments, to arrange support of the project. Some of these departments include training to support deployment, document preparation and production, quality assurance, finance, and logistics. Other umbrella tasks include determining how to reuse working software, how to conduct formal reviews and approve project documents and planning for risks. Throughout the project, performance should be objectively measured, resulting in quantitative data. This data can be compared against established metrics to evaluate the project status, and make course corrections to keep the project on track.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

SDLC Umbrella Tasks

Learning Objective

*After completing this topic, you should be able to*

* *use the Software Development Lifecycle (SDLC) Umbrella Tasks in Software Practices (Waterfall)*

**1. Umbrella tasks in software practices**

SDLC umbrella tasks are defined by activities, not clearly defined work phases. The project management plan guides how the umbrella tasks are implemented in the project. Software project management includes a wide range of activities, that keep the project manager very busy. Detailed schedules must be created and tracked along with current cash flow, to stay within the approved budget. Software releases to QA and production must be managed and controlled, and the project manager, must coordinate activities with many project infrastructure and support groups. The project will need development environments and data management controlled by the IT department, and office space and furniture from facilities. Finance controls the project's cash flow, and senior management monitors the project's progress through status reports and reviews. Human Resources facilitates the ramp-up of skilled workers for the development team. And marketing, training, documentation, and logistics, are required to transform the software application into a deployable product. A project is controlled and evaluated through a series of formal technical reviews. Reviews are used to formally accept artifacts, signifying a phase of the SDLC is complete.   
*Heading: SDLC Umbrella Tasks.  
  
SDLC Umbrella task activities include:  
  
Software Project Management.  
  
This includes schedule tracking, budget management, release management, and coordination of activities with external departments.  
  
Formal Technical Reviews.  
  
These control and evaluate the project. They include change management reviews and project milestone documents.*   
  
These artifacts are normally a document, such as the requirements, design, test plan or test documents. They may also include diagrams, laying out the system or database design. Review participants are qualified team members and stakeholders who determine the completeness, accuracy and suitability of a project artifact. Stakeholders may hold additional formal reviews to control project scope and cost. One example is change management reviews, to investigate and improve a requested requirement or scope change. Software quality assurance provides oversight across the entire software development process, to ensure requirements for the project are fulfilled and the software produced is of high quality. QA is responsible for promoting standards and best practices, as well as ensuring the project adheres to industry regulations, and business procedures and processes. QA oversight encompasses all software tasks and phases, but the main focus is evaluating and validating the resulting software application. This evaluation focuses on two major areas: functional requirements specification compliance and non-functional requirements such as robustness, performance, and maintainability.   
*Heading: SDLC Umbrella Tasks.  
  
Software Quality Assurance.  
  
QA activities evaluate functional requirements specification compliance as well as non-functional requirements, and promotes best practices.*   
  
Software configuration management is concerned with how the project will implement, and control the aspects of software archival, update, Build and versioning. This includes the choice of a CM System, which provides archival and storage of not only the software developed, but also the associated test code, build files, configuration, and documentation. CM systems provide a control change process for software, which logs modifications made to the code base, when they were made and who made them. The CM system enables the project to use revision controls to establish baselines, ensuring a repeatable Build process for released code. CM systems can also be linked to the defect tracking system, to tie code modifications to specific defects. Reusability management is the process of identifying software objects, functions, components and modules, which can be reused in other software modules or projects with little or no modification.   
  
Reusing existing software can significantly reduce project effort and cost, in both the development and test phases. However, a successful software reuse effort requires buy-in from both management and development. Management must give the developers the extra time needed to create reusable software components. And the software developers must keep reusability in mind, when designing and developing software. Software reusability affects multiple areas, including build management, packaging, distribution, deployment, and installation of the reusable components. And a reusable component in production needs special consideration when dealing with configuration, maintenance, and upgrades, to ensure modifications do not affect other areas where the component is being used. Over the life of the project, the project manager must gather objective quantitative measurement data, and analyze it to influence future actions and plans.   
  
The data gathered can be compared to metrics around schedule, size, complexity, cost and quality to evaluate the current project status with respect to the project plan. This analysis allows the project manager to characterize the current processes and environment, identifying the root causes of issues and inefficiencies, and making the necessary changes to keep the project on course. Software measurements and metrics influence planning for the schedule and budget and drives software development optimization. Documentation involves several areas and departments. The development and QA teams will prepare their own SDLC project documents, including the requirements, design and test plans. These will have their own archival and control processes, normally in the SCM system. Documentation used in production are produced by technical writers, they include the user manuals, which describe the software's interfaces and features for end users and the support manuals, for the operators and administrators. Marketing will produce its own documentation used to sell the software, including brochures, white papers, and promotional material.   
  
Once documentation is written, there must be a process to review and accept it, then produce multiple copies to accompany the software product. Risk management is an ongoing activity in any project, which attempts to identify, measure, and assess risks, then develop strategies to mitigate or manage the risks. For each risk identified, analysis is performed to determine the possibility of the risk occurring, and its effect on the project. Risks are associated with data that is objective, timely and accurate. This associated data is used to identify and document thresholds, which will act as indicators that a risk is occurring. Once the risk plan is established, it must be used to monitor the project data for the predetermined data thresholds. When the metric risk indicators pass the predetermined threshold, it triggers the project to invoke the strategies developed to handle the risk.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

SDLC Analysis Phase

Learning Objective

*After completing this topic, you should be able to*

* *use the SDLC Analysis phase of Waterfall*

**1. The Analysis phase**

The Analysis phase is used to dig deep into the system concept, to completely understand the system proposed and determine what needs to be implemented, to satisfy the concept. This is done by reviewing all the analysis and research from the Startup phases, to flesh out the features and purpose of the system. Additionally, interviews are held with users and product management to produce use cases. Use cases document scenarios, or sequence of events, capturing user input to the system, system behavior, and possible outputs. Use cases explore main user paths, error handling and recovery, and possible system issues. They capture how a system should interact with users to satisfy desired functionality, which helps to identify and clarify requirements. Use cases are an excellent way to uncover and document customer and team expectation, which may not be obvious. Once the system is understood, it can be broken down into smaller components and subsystems. This breakdown gives a high-level system architecture and aids in documenting the functions and operation of the final system.   
*Heading: SDLC Analysis Phase Activities.  
  
This is the most critical phase for success of the project.  
  
The Analysis phase is the first phase of the SDLC and it feeds into the Design phase.  
  
System Breakdown into Components.  
  
The system is broken down into components in order to understand the problem, define project goals, and build use cases.   
  
The players include stakeholders and development team leaders.  
  
Functional and System Requirements.  
  
The functional and system requirements are identified to determine what needs to be implemented to deliver the desired features and functions, and meet customer expectation.*   
  
This work is done by the development team leaders, including the system engineers, architects, project manager, and small team leads. The development team gathers information from the product manager who relays input from the customers and stakeholders. Additional input must be gathered from company authorities in security, compliance, and IT operations, to ensure all external requirements dealing with regulations, privacy, or company processes and standards, are accommodated. The goal of the Analysis phase is to identify the system's functional and system requirements, and the high level system architecture. These together describe in detail exactly what needs to be implemented in the final product, to deliver the desired features and functions. The Analysis phase produces a number of artifacts, that guide the Design phase. The most important artifact is the requirements document, which is often used as a contractual document of what the project will deliver. The requirements document holds functional and system requirements, which determine what needs to be implemented in the final product.   
*Heading: SDLC Analysis Phase Artifacts.  
  
Requirements Document.  
  
This describes what must be delivered in terms of functional requirements as well as system requirements. Each requirements must be testable.*   
  
Each requirement defines some aspect of the system's functions, behavior, performance, or interfaces. Requirements only state what the system must do, they never say how a feature will be implemented and each requirement must be testable, so that it can be verified during validation. A general requirement like the system must be user friendly, relays a desirable feature that cannot be quantitatively measured, so it is not testable and cannot be verified. Testability is important because the approved requirements document is contractual. It is common to trace each requirement through design, to at least one test case to ensure all requirements are implemented and validated. Technical and business groups must work together to uncover the system requirements. Communication between departments is facilitated by graphs, pictures, and diagrams that help departments with different thought processes converge on a single perspective.   
  
Use cases show the interaction between system users and the functions the system must perform. Use cases enable teams to explore system behavior, fleshing out the boundaries of the system, the different perspectives of different user roles, and uncovering errors and the correct recovery behavior. Interviews with stakeholders and users uncover goals and expectations, which are critical to the success of the project. These interviews can be held one-on-one, in focus groups, or by sitting with the user and observing how they currently do their work. The notes from these interviews are vital input to writing the requirements. A high-level conceptual model of the database is often constructed, showing entities and the relationships between them. These entity relationship diagrams help understand the data items that will be used and stored by the final system.   
*Heading: SDLC Analysis Phase Artifacts.  
  
Working documents include use cases, interview notes, a database conceptual model, and system architecture diagrams.*   
  
The conceptual entity relationship model will be fleshed out into a logical model adding details like, attributes, identifiers, and cardinalities, as everyone comes to agreement on the entities and their relationships. System architecture diagrams show the different physical computers and devices, that will work together to create the software system, as well as the software's interaction with external systems and data stores. These diagrams, which were created during the Planning phase, are used to understand how data flows through the system. The functions performed by each server or device is broken down into components and subsystems to create a rough high-level software architecture. As each requirement is discovered, it can be associated with one or more of these software components and subsystems, to help categorize and organize the requirements. These categories often become the sections of the final requirements document.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

SDLC Design Phase

Learning Objective

*After completing this topic, you should be able to*

* *implement the Waterfall SDLC Design phase*

**1. The Design phase**

The Design phase is where the requirements specifying what the system must do, are translated into the design, which specifies how the system will be implemented. The design is repeatedly refined, adding more and more detail, until it becomes a recipe for how to implement and test the final system. This work is done by the entire development team, including the architect, database and software engineers, and the quality assurance team, all guided by project management. The process starts with refining the high-level software architecture, defining the high-level interfaces between the modules. The high-level interface description shows which modules talk to each other and what is sent between them. This includes not only communication between modules, but also communication to existing systems in the production environment. The description will also specify the method or protocol used for the communication, and the physical network for distributed systems. The software architecture will be reviewed to ensure it covers all functional requirements and will deliver a product that meets the system requirements as well, such as performance, security, and scalability.   
*Heading: SDLC Design Phase Activities.  
  
The Design phase happens after the Analysis phase and feeds into the Development phase.  
  
Software Design.  
  
Software design specifies how to implement requirements and describe the system in detail.   
  
The players include the entire Development team and quality assurance.  
  
Design Activities.  
  
These activities are used to define the high level design, produce detailed module designs, and implement quality assurance planning.*   
  
The different modules in the software architecture will be assigned to members on the development team, who will be responsible for their detailed design. During detailed design, large modules would be broken down into smaller ones, with their own descriptions, interfaces and diagrams, which show how to implement the module and software. design engineers often use prototyping, investigating areas that are unclear or unknown to gain clarity and determine a design approach. Stakeholders and customers review and comment on GUI wireframes and screen layouts that show the flow to the product’s user interface. And database designers add additional detail to the logical entity relationship model, to create the final database schema. Quality Assurance is also hard at work, using their requirements to develop the strategy the QA team will use to validate the final system. Areas of QA planning will include how defects will be reported and tracked, how software will be delivered from development to the QA team for validation, and how the different tests will be performed.   
  
Planning also includes specifications for the isolated test environments. An area of special interest is how acceptance testing with the customer will be accomplished, as this determines if the customer will accept the implemented system. During this phase, build out of the development environments must be completed, as they will be needed to implement the application in the Development phase that comes next. The Design phase activities result in artifacts that are used in the Development phase, to implement the final system. The architecture, or high level design document, gives a blueprint of the entire system, which will be built. It will include the software architecture which specifies the modules and the interfaces between them and external systems, as well as communication methods. The user interface flow will be described with wireframe screens and the logical database model will also be included, showing entities and their relationships. The architecture document will include many diagrams such as UML system diagrams and database entity relationship diagrams. However, it will still primarily specify what must be implemented, without details of how to implement it.   
*Heading: SDLC Design Phase Artifacts.  
  
Architecture Design Document.  
  
This is used to identify all modules – components and subsystems, describe system interfaces, user interface wireframes, and a database logical module.*   
  
The details of how to implement each module are specified in the detailed design document. For each module, the detailed design document describes the algorithms and data structures used, detailed design diagrams, and detailed interfaces. The interfaces will include the exact data items transferred between modules, along with their data type and other attributes, like permanence and scope. Detailed design diagrams may include UML diagrams that show the structure, and behavior of objects in the module, dataflow diagrams, flowcharts, and sequence diagrams. Some projects will use pseudo-code to define the actual structure of the code, which will be implemented for each module. The user interface will be mocked-up to demonstrate the look, and feel, of the final screens the user will experience. And the database detailed design will be created, which specifies the details of each table, including column names and data types, as well as column and table constraints. Views, triggers, stored procedures, and other database objects will also be specified in detail. As well as the method of how users will be authenticated, and what database objects will be authorized for the different roles. The resulting detailed database design will be used to write the database schema.   
*Heading: SDLC Design Phase Artifacts.  
  
The Detailed Design Document.   
  
This describes the design of each module. It documents details of interfaces, detailed user interface mock-ups, and database detailed design.*   
  
In distributed systems, the data dictionary will also be created, to specify the meaning, origin, usage, and format of each data item, and the relationships between it, and other data items in the production environment. All the planning done by the QA team is captured in the quality assurance plan document. The quality assurance plan describes how the validation strategy will be implemented to verify the system and ensure it is of high quality. The quality assurance plan will include a schedule of the QA activities throughout the project, and the documents that will be produced in each project phase. The review process for formal reviews will also be described. As well as any metrics which will be collected. The plan will also include guidelines to ensure adherence to standards, regulations, and company business processes. A large part of the quality assurance plan will cover the types of tests performed, including unit, functional compliance, system, regression, and acceptance testing. The different test environments required will be described in detail, as well as the methods used to report and track defects. As QA has responsibility for oversight across the entire software development process, other areas may also be covered to ensure quality and training, deployment, and audits.   
*Heading: SDLC Design Phase Artifacts.  
  
The Quality Assurance Plan.  
  
This is a strategy for verifying the final product and includes the types of tests performed, test coverage of requirement, test methods, and reporting strategy, as well as test environment specifications.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

SDLC Implementation Phase

Learning Objective

*After completing this topic, you should be able to*

* *describe the Implementation phase of software development in Waterfall*

**1. The Implementation phase**

In the Development phase, the software engineers use the design documents and diagrams to guide them in implementing the software application. This work is done by the entire development team guided by the project manager. Each engineer begins work on their module in their own private development environment. Each module is coded, compiled, and unit tested to ensure it meets the requirements assigned to the module. Unit tests are small standalone applications, which exercise a unit of code, invoking the code's methods and functions and monitoring the code's behavior. The unit test may also simulate external methods and objects the code expects to make the code act as if it was in a complete system. Unit tests are used to verify each unit of code is correct, before it is integrated into the system. Code reviews with other developers may also be required to catch bugs early before system integration. Build scripts are used to control compilation and linking of the software, into executable components and subsystems. The build scripts ensure the software is always built the same way, no matter who is building the system.   
*Heading: SDLC Implementation Phase Activities.  
  
This is the longest phase in the SDLC.  
  
Implementation happens in the Development phase, which occurs after Design, and feeds into the Validation phase.  
  
Implement the System.  
  
Individual modules are assigned to each engineer to write the source code and perform unit testing. Testing include system integration testing.   
  
The players include the development team and quality assurance.  
  
Implementation Environments.  
  
These include the development environment, build environment, and system integration environment.*   
  
Developer source files are added to the scripts as they are completed, so they can all be built together as a single combined system. Many times daily builds are automated to find build issues early. These builds take place in the build environment, which has the approved revisions of all the build tools and software installed on suitable hardware. A best practice is for developers to update their individual development environments, with the latest build modules often to find integration issues as early as possible. As the software modules are completed, they are integrated with other modules in the system integration environment. Merging the individual software units of code into a single system allows developers to test the system as a whole, ensuring the individual units of code work together. The system integration environment will contain a copy of the system databases, populated with test data.   
  
Integration testing by developers is not rigidly controlled, system integration environments may have mocked up modules installed to simulate external systems or a software that is not yet complete. Individual developers normally concentrate testing on their own modules, integrated with the rest of the system instead of exercising the system with a comprehensive set of tests that cover all the requirements. Besides testing the system features, the system integration environment can be used to do performance and stress testing to ensure the system can meet system level agreement requirements. Software that has been built, integrated and tested as a single system by the development team is sometimes known as engineering software, because it has not yet been independently verified by quality assurance. While the developers are implementing the software, the QA team is busy preparing for the validation phase, by setting up the test environments and determining how to verify each requirement in a test case.   
  
The software code, unit test, build scripts, database schema and configuration files are all stored in the software configuration management system, which controls access and tracks changes. The SCM system uses authentication methods to control access to the files of archives. Authorization schemes are used to restrict the files a user can access to those for their own project. This enables a single SCM system to securely archive files from multiple projects. The SCM system also keeps track of revisions to the source files, tracking the changes that were made, who made them and when they were made. And it can be used to create named baselines, which capture the state of the code at a particular point in time. Baselines identify the exact code revisions, which were used to build an application delivered to the QA team or the customer. This ensures an exact copy of a baseline application that can be built and delivered from the files archived in the SCM system.   
*Heading: SDLC Implementation Phase Artifacts.  
  
The Engineering Source Code.  
  
Related files are archived and controlled in the SCM system. It is integrated into the system and includes unit tests and databases with populated test data.*   
  
Any documentation done by the development team, during the implementation phase is contained in the source code itself, in the form of comments. Comments are meant to be used by developers as they are reading the code to identify modules and subsystems, explain algorithms, state the file's author and revision history and include a copyright statement, identifying the company, which owns the code. The implementation phase artifacts delivered by the development team are the files archived in the SCM system along with the SCM tracking history and baselines. The source code will have been coded and tested as well as system integration tested as a complete working system. A uniquely identified baseline will be created to mark the software delivered to QA for validation. Building the engineering software for delivery to QA, entails pulling all the files from the SCM system tagged with the baseline identifier, then using the build scripts which were pulled to build the system executables. These executables can be packaged in an installation program which is delivered to QA.   
  
The QA team's work during the implementation phase, results in the isolated test environments with all the required hardware and software for validating the engineering code. The development team is typically not given access to the test environment. To ensure the only code being tested is the engineering code built with the baseline identifier. The QA test engineers will also produce a group of test cases. Each test case is a recipe for testing a feature or aspect of the application. It will include the steps necessary to set up the test, putting the environment in a known state and steps to perform the test and the expected result. Each test case will trace back to one or more requirements. If the application passes the test, it is said to comply with the associated requirement.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

SDLC Validation Phase

Learning Objective

*After completing this topic, you should be able to*

* *use the Validation phase in Waterfall*

**1. The Validation phase**

The Validation phase begins when the engineering software is delivered to the QA test engineers, as a complete working application. The delivered application is tested in an isolated environment, where all variables which could influence the test, can be controlled. Development does not have access to the test environment, so the only way to get code installed on it, is to formally deliver it to the QA team. QA testing is performed on the entire application, interacting with it as the end user will, once it reaches production. QA testers will run several different types of test cases. The majority of test cases will ensure the software satisfies the functional and system requirements. Functional test cases demonstrate the features and behavior of the system. System requirements measure system aspects, such as the application's performance, scalability, security, or reliability. Compatibility test cases may be needed to ensure the application interacts correctly with existing systems, or will run on all supported versions of an operating system. Each test case should trace back to one or more requirements, and all requirements should be covered by at least one test case. The software is often delivered with an installation application, so the first test case is that the software installs correctly and the application can be started.   
*Heading: SDLC Validation Phase Activities.  
  
Validation phase activities ensure that code satisfies requirements.  
  
Validation occurs after the Development phase and feeds into the Deployment phase.  
  
Execute Test Cases.  
  
These ensure the system meets the functional and system requirements, as well as testing user acceptance.   
  
The players include QA test engineers and entire development team for bug fixes.  
  
System Test Environment.  
  
The environment is isolated and highly controlled, the entire system is delivered with installation, and is installed in a clean environment.*   
  
Once the application is running, the test engineers can begin to execute the test cases. Executing a test case is a rigid process, where the environment must first be set in a known state detailed in the test case. This could include loading test data onto the database from backups or ensuring multiple external systems are running and ready to communicate with the application. Then the test case steps are executed in the order defined. Each step will include some action from the tester, with defined inputs. The expected outputs will be listed, which the tester will compare against the behavior of the application. If the behavior matches the expected result, the test passes. Otherwise, the test fails and a defect is recorded for the development team to remedy. The development team will modify the software to fix the defects as they are received. When enough bugs have been fixed to warrant a new version of the software, a new baseline is established and used to build a new revision, which is delivered to the QA team. Once the new software is installed, the testers must do a round of regression testing. Re-running all previous tests to ensure the modifications did not break working code. When a previously recorded defect no longer occurs in the new revision of the code, the QA team closes the defect.   
  
At any time, the number of open defects is a metric that indicates the stability of the application, and how well it satisfies the requirements. When the number of open defects reaches a threshold, indicating the software is ready for production, the QA team conducts user acceptance testing with the customer. If the customer agrees the software satisfies the requirements, they sign off on the project, and the software can move to deployment. QA records its finding from validation in an artifact, called the test analysis report. Each test case executed will be documented in the report, with the description of the system's capabilities tested, an analysis of the system's behavior, and the test results, either passed or fail. The test analysis report also presents the defects which are still open, to show where the software is deficient, along with a description of the overall effects of the open defects on the readiness of the software for production. Any differences between the test environment and the final production environment will be documented. And the conclusion will give the QA team's assessment of the state of the software, and a recommendation on whether or not to move forward.   
*Heading: SDLC Validation Phase Artifacts.  
  
Test Analysis Reports.  
  
The reports include test case results indicating pass or fail, as well as a list of prioritized open defects.*   
  
The other artifact produced in the Validation phase, is the validated software which is ready for production. The process of transforming engineering software into production software is cyclic. As QA finds bugs in the delivered software during testing, it records them in the defect tracking system. The Development team must fix the open bugs and create a new build of the software, which is delivered to QA for another round of testing. Several rounds of testing may overlap with development, allowing QA to test the software as it is being written. When a new revision of the code is delivered, regression testing happens, to ensure working features were not broken in the updated software. This cycle is then repeated until a stable application is produced, which is ready for user acceptance testing with the customer. When the customer signs off on the validated software, it is ready to go to deployment.   
*Heading: SDLC Validation Phase Artifacts.  
  
Source Code Revisions,  
  
Revisions provide the entire system with bug fix updates and a list of the defects that have been closed. Each modification is tied to a specific defect number.  
  
A diagram illustrates how the validation process is cyclical. The updated revision is sent to Validation where it is tested. If unresolved bugs are found, the software is sent back to Development to implement bug fix updates.   
  
This cycle repeats until the system is validated and can proceed to Deployment.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

SDLC Deployment Phase

Learning Objective

*After completing this topic, you should be able to*

* *implement the Deployment phase of Waterfall*

**1. The Deployment phase**

In the Deployment phase, the software is transformed into a product, and installed in the production environment, where it will be used by the customer. The transformation of validated engineering code into production software, takes the work and support of many company departments. Documentation must produce manuals, and Training must create classes to support users and administrators. If the software is a product that will be sold, Logistics must arrange for packaging and shipment, and Marketing must set up the supply chain and produce the materials used to market, and sell the product. As the main work of the software developers and testers is done, they will start to transition to new projects, which are ramping up. But development team leads, most knowledgeable in all aspects of the system, will remain involved to troubleshoot issues found in early deployment, assist in training and documentation efforts, and help transition the software to the new support team. The support team may consist of administrators, operators, professional services, and helpdesk workers, who will support customers using the system. Often internal and external acceptance testing will be conducted with small focus groups, to flush out any issues before full production.   
*Heading: SDLC Deployment Phase Activities.  
  
This is the phase where a product is put into production. It is the final phase in the lifecycle and receives input from the Validation phase.  
  
Transform Software into a Product.  
  
This involves a number of players including end users and administrators, development team leads, and many company departments.  
  
Supporting documents are created, internal and external acceptance testing is performed, and user training is carried out.*   
  
Alpha tests are known as internal acceptance testing, conducted in-house at the developer's site. The testers consist of small groups of customers, interested in the release, or an independent test team. An environment is setup to simulate the production environment, where alpha testers can interact with the software. Beta tests are known as external user acceptance testing, where the software is released to a limited group of customers to test in their own environment. The developers and QA are heavily involved during alpha and beta testing, to document, fix, and verify any bugs found. As defects are closed, new releases may be created and delivered, along with a list of bugs closed in the new release. This cycle of finding defects, fixing them, and issuing new releases, may repeat multiple times during testing. When all rounds of acceptance testing are complete, the software will be deployed into production. There are two types of production software, each with its own different production environment and support needs.   
  
A managed system is installed in a company's IT environment and accessed by many users. This could be an internal facing system, like a helpdesk tracking application, or a customer resource management system. Or it could be an external facing system, like a company's website used to perform e-commerce with customers. Managed systems are used to conduct a company's business and require many resources to help them run smoothly. Logistic needs include space for the equipment, with adequate cooling, power, and networking. As well as the servers and supporting software licenses for operating systems, databases, and application servers. Other resources include highly-skilled personnel, dedicated to administration and keeping the system operational. Production software that will be sold as an off-the-shelf software product, requires more packaging than managed systems. But they do not require the company provide an operational environment, since the product will be installed at the customer's site.   
*Heading: SDLC Deployment Phase Activities.  
  
Deploy in Production Environment.  
  
There are two types of production environments: the managed system and off the shelf software.*   
  
To support external customers, the company will need a helpdesk call center with trained staff, and customer facing resources like a website with information about the product and the ability to download updates and supporting software, like software drivers. In the case of large enterprise system products, like a DBMS or CRM system, professional services could be deployed to assist customers in setting up and customizing their system. Production software is accompanied by many artifacts, which help transform it from validated code into a product. It must have an installation script, or application, to aid in installing and configuring the software in the production environment. Installation may be guided by readme files, installation instructions, and quick start flyers. And the software should have a user's manual, explaining how to use the features and functions of the system. A large enterprise managed system, will also be accompanied by operation and maintenance support documents, telling the support personnel how to configure and administer the running system. This could also include instructions on how to configure external systems to communicate with the software, such as loading the application schema onto an existing DBMS database.   
*Heading: SDLC Deployment Phase Artifacts.  
  
Final Product.  
  
This is accompanied by a number of artifacts including the installation application, validated code and supporting configuration, DBMS with database, user documentation such as user and administration manuals and marketing documentation, and product packaging.*   
  
Marketing artifacts will include brochures and promotional material to handout at conferences and stores. Complex systems may be associated with white papers, describing the issues the product solves. Marketing is also heavily involved in the design of the product packaging, including the box design and enclosed flyers, with information on warranties, installation codes, and promotions. And the method of transporting the software to the customer needs to be implemented. Whether it's a website for download, or a portable media like a DVD. The production of training classes and material is crucial to a smooth deployment. Clear, concise training material and classes help end users feel comfortable with the new software, enabling them to quickly come up to speed and become productive. Helpdesk workers need training classes and quick help material in order to support customers who are learning the new software. And in enterprise managed systems, operator and administrator training is critical in keeping the software operational, so it can continue to run the company's business.   
*Heading: SDLC Deployment Phase Artifacts.  
  
Training.  
  
This included user training as well as support training.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

SDLC Cost of Change

Learning Objective

*After completing this topic, you should be able to*

* *calculate the cost of change using a Waterfall software development model*

**1. The cost of change**

The cost of finding and fixing a defect in the requirements increases exponentially as the project progresses. One reason for this increased cost is as the project progresses, more and more artifacts are reviewed and accepted. Changes to the requirements means all completed artifacts must be updated, reviewed, and accepted again to accommodate the change. And the number of people involved in creating, updating, and reviewing the artifacts increases as the project progresses. Software is especially expensive to modify. If a software design does not satisfy its requirements, it is easy to fix while in the design stage, when all that needs updating is the design diagrams and document. However, if the error is not realized, until the software components are being integrated, then unit tested code must be modified along with the unit tests. This could include modules that interface with the defective code, causing work to halt until the rework is complete. In severe cases, the design defect could require changes to the software architecture, affecting large portions of the system.   
*Heading: Dynamics Software Projects Environment.  
  
Software Projects are Dynamic.  
  
Changing requirements are common and the cost increases exponentially as the project progresses. This is because of the need to update accepted artifacts from previous phases and more players become involved as the project progresses.  
  
A graph on the slide shows the SDLC environments along the X-axis and cost on the Y-axis. The curve on the graph starts are the origin and it increases exponentially as it passes from one phase to the next.  
  
Project Management Always Affected.  
  
Delays in schedule could occur. These could lead to increased costs and the need for more budget.*   
  
These types of changes found late in the project, delay the scheduled due dates. Project management must then push out the schedule to accommodate the rework and work with stakeholders to increase the budget to pay for it. Delays in the schedule affect release dates, set by product management and marketing in coordination with departments supporting deployment. Let's explore the rising cost of change to the requirements as the project moves through the SDLC phases. The product manager normally notices a defect in the requirements while interacting with the stakeholders. Stakeholders could be customers or high-level executives sponsoring the project. If a requirement change is found in analysis, it only involves the architect writing the requirements document. The architect will discuss the change with the product manager and update the requirements document before it is reviewed and accepted by the stakeholders. If a requirements change is found during design, the design engineers and architect will need to review the change with the product manager to understand how it affects the design.   
*Heading: Cost of Changing Requirement.  
  
A graph shows the SDLC environments along the X-axis. The X-axis is split into five columns: Requirements, Design, Coding, Testing, and Deployment.  
  
A concave curve labeled Cost starts at the origin and increases exponentially as it crosses from one phase to the next.  
  
An icon labeled Product manager is placed at the top of the Y-axis outside of the graph, the arrow leading from the icon spans all of the columns.  
  
An icon labeled Architect is placed in the Requirements column. The arrow leading from the icon spans the Design, Coding, Testing, and, Deployment environments.  
  
Below the graph, under the Requirements column, and next to the folders icon is shows Requirements Document.  
  
An icon labeled S/W Developers is placed in the Design column, the arrow leading from it spans the Coding, Testing, and Deployment environments.  
  
Below the graph, under the Design column, a label reads High Level and Detailed Design Document.*   
  
The requirements will need to be updated to accommodate the change and then must be re-reviewed and accepted by the stakeholders. And the design will be modified to accommodate the updated requirement. This design change may involve several modules and the developers working on them. If a requirement change is found during implementation, the development team and architect will need to review the change with the product manager to understand how it affects the design and the code already developed. The requirements and design will need to be updated to accommodate the change and then both must be re-reviewed and accepted by the stakeholders. Changes to the design will mean modifications to the code already implemented, as well as the associated unit tests. And the QA test cases will need to be modified to accommodate testing the requirement change. Changes this late in the project may require pushing back the scheduled release of the software, affecting all departments involved.   
*An icon labeled QA Engineers is placed across the Coding and Testing columns. The arrow leading from it spans the Testing and Deployment environments.  
  
Below the graph, under the Coding column, the label Source Code Test Cases is displayed. Under the Testing column it shows New Tests and Regression Testing.*   
  
If a requirements change is found during validation, the test engineers validating the system, development team, and architect will need to review the change for the product manager to understand how it affects the design, integrated code, and test cases. The requirements and the design will need to be updated to accommodate change and then both must be re-reviewed and accepted by the stakeholders. The integrated code will need to be modified which could affect multiple modules working together in the system. Code changes mean updates to the unit test and QA test cases and could entail design and code reviews by the rest of the development team, to minimize the effect on the existing code base. QA will need to perform a full regression test of all previously passed test cases to ensure the changes did not affect working code.   
  
If a requirements change is found during deployment, all departments involved with the development and deployment of the software are affected. Again all artifacts produced in previous phases, must be updated and re-reviewed and validated software goes back to rework the design and update the code. The updated software must be validated, including a full regression test of all the test cases to ensure the changes did not affect working code. Schedule changes will affect not only development and QA, but also support departments, including training, documentation, product packaging, logistics and marketing, as well as customers, participating in alpha and beta tests or expecting delivery of production software. And developers that had already transitioned to new projects may need to be called back, affecting the new project as well. If a requirements change is found, when the software is in production, during the operation and maintenance phase, then an entire new project must be approved, financed and executed through all phases of the SDLC to implement the change in the production product.   
*An icon labeled Customers and Support Departments is placed in the Deployment column. Under the Deployment column, the label is: Training, Documentation, User and Support Manual, Product Packaging.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

The Sequential Approach Software Method

Learning Objective

*After completing this topic, you should be able to*

* *use the sequential approach to software modeling in Waterfall*

**1. The sequential approach**

The two broad approaches to how software models implement all phases of the Software Development Lifecycle are the sequential and the iterative approach. Both have their pros and cons. The sequential software method advocates an orderly linear approach to software development, where each phase is visited only once and all downstream phases must depend on the work done in previous phases being complete and correct. In the sequential software methods, each phase of the Software Development Lifecycle must be completed before the next phase has begun. The project flow from one phase to the next is controlled by gates between the phases. Each gate documents the work that must be completed in the previous phase before the project can advance to the next phase. The work is represented by artifacts, produced during the phase; such as documents, which must be formally reviewed and accepted. Once all work specified by the gate is approved, the project is allowed to progress through the gate to the next phase.   
*Heading: Software Methods: The Sequential Approach.  
  
Linear Process.  
  
The process involves the rigid adherence to phases, is highly controlled, and relies on a predictive software model.  
  
There are three common sequential approaches: Waterfall, V-Model, and Sashimi.  
  
A illustration shows three stages: Stage 1: Plan; Stage 2: Develop; and Stage 3: Rollout.  
  
Each stage consists of activities. The activities are grouped and placed in sequence horizontally. Stage 1 activities include scope definition and requirement analysis. Stage 2 activities include architecture design, coding, and testing. Stage 3 activities include user acceptance, production readiness.  
  
Each stage is separated by a numbered gate. Gate 1 is labeled Budget approval and separates Stage 1 and Stage 2.  
  
Gate 2 is labeled Validation completion and separates Stage 2 and Stage 3. Gate 3 is labeled launch and is placed after Stage 3.*   
  
All project movement is forward toward the next gate and phase. Previous phases cannot be revisited. This linear approach results in a predictive software method, which focuses on detailed analysis and planning all aspects of the project's future in detail. This causes rigidity when faced with changing requirements later in the project. Many times, a formal Change Control Board is instituted to review any changes to the requirements and ensure only valuable changes are considered. With sequential software methods, business value is only delivered at the very end of the project, when the customer sees the complete system for the first time. Common software modules that implement a sequential approach, include the Waterfall model, V-Model, and Sashimi. Sequential approaches were the predominant software development method for many years and are still in use today.   
  
This strict adherence to the Software Development Lifecycle phases make them easy to understand and also easy to use to control and monitor software projects. Sequential models provide structure, which is a benefit when a project has inexperienced staff. Each phase has set functions and responsibilities which help management plan, staff and track the different phases of the project. And milestones are well understood, giving a measure of how the project is progressing. Lastly, sequential methods enhance requirement stability as it is very difficult to change a requirement once the Analysis phase is passed and the requirement document is approved. Sequential methods, with their rigid adherence to forward movement through each phase, dictate all requirements must be known upfront. Since a change found earlier in the stages of a project are cheaper to accommodate, than the same change found later in the project. The extra time spent planning could benefit a project using a sequential approach.   
*Heading: Sequential Approach Pros and Cons.  
  
Pros include that it provides an easy to understand structure with well understood milestones.  
  
Cons include that the approach is not adaptive to changing requirements, it dictates that all requirements must be known up front, it gives a false impression of control and progress, the customer must wait until the end of the project for results, and customer expectation may not match the result.*   
  
However, software development almost always involves changing requirements and goals while the project is in progress. It is rare to know all the requirements upfront. Since the deliverables created for each phase are considered frozen, the sequential method inhibits flexibility and does not reflect the iterative problem-solving nature of software development. The well-known structure of sequential methods, with their formal reviews and approval process, can give management a false impression of progress and control. The project may seem to be running smoothly, but since integration and testing is done at the very end of the project, the customer has limited opportunity to preview the system until it is already complete. If the requirements did not capture the customer's expectation or their expectations changed as the project progressed, the customer may not accept the final product, leading to a failed project.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

Sequential Approach Software Models

Learning Objective

*After completing this topic, you should be able to*

* *use the different sequential approach software models*

**1. The sequential approach models**

The Waterfall software model rigidly follows the Software Development Lifecycle phases, allowing the project to be in only one phase at a time. All the work in the current phase must be completed, reviewed, and approved, before the project is allowed to advance to the next phase. And a phase, once completed is not revisited. All the artifacts produced during the completed phase are frozen. The Waterfall method complements a rigid management structure, where all the requirements are documented at the beginning of the project, along with an agreed-upon schedule and budget. The requirements, budget, and schedule, are normally part of the contract and are very hard to change once the project is underway. Waterfall's close adherence to the SDLC phases, makes it easy to understand. Everyone associated with the team knows what phase the project is in, and exactly what must be accomplished to complete the phase. However, since all requirements must be known up front, the Waterfall model does not have a process to accommodate changing requirements, making it inflexible.   
*Heading: The Waterfall Software Model.  
  
Waterfall rigidly follows SDLC phases. Gates control movement to the next phase and the completed phases cannot be revisited.  
  
The business value is delivered at the end of a project. It is easy to understand and control but is inflexible with changing requirements.  
  
Waterfall is a good fit when the requirements and technology are well understood, and the product definition is stable.  
  
The Waterfall model has five steps that are sequential. They are Requirements or Analysis, Design, Coding, Testing, and Maintenance.*   
  
This can be especially problematic, since the customer does not see the product till it goes through acceptance testing, at the very end of the project. In projects with long durations, the customer's expectations, state-of-the-art technology, or product environment could have evolved to the point that the software is obsolete before it is completed. The Waterfall method works best when the requirements and technology employed in the software are very well known, clearly stated, and static. These types of projects are typically short in duration with a clear staple product definition, such as creating a new version of an existing product by adding new features, or porting the product to a new platform. The Sashimi software model is very similar to Waterfall, also following the SDLC phases. However Sashimi allows the phases to overlap a bit, in much the same manner as fish slices overlap in plated Sashimi.   
*Heading: Sashimi and V-Model Software Model.  
  
The Sashimi Waterfall model is similar to the Waterfall model but with overlapping phases.   
  
The overlapping adds iteration to adjourning phases, which encourages continuity between phases and results in early defect detection and minimizes rework.*   
  
The Sashimi model dictates that no phase can be completed, until the next phase is at least partially explored. For example implementation starts before design is complete, so issues with the design can be uncovered early and addressed. This adds iteration to adjourning phases, adding some flexibility to the process. Rework is minimized, because the work of the previous phase has not yet been approved and frozen, allowing updates as defects are discovered in the next phase. The Sashimi process is most appropriate for medium-sized projects, with effective informal communication, between the different project roles. This allows insights gained in one phase to be communicated to a previous phase, influencing the artifacts produced, which guide the final product. The V-Model also follows the SDLC phases, but the activities in each phase put an emphasis on verification and validation. The phases are arranged with the project definition phase as going down the left-hand side of the V, and the project test and integration phase is going up the right-hand side. The development phase forms the V's base.   
*Heading: Sashimi and V-Model Software Model.  
  
The illustration shows how, in the Sashimi model, there is a constant exchange of information happening between the current phase and the next phase, rather than a fixed sequential movement from one phase to the next like with the Waterfall model.  
  
V-Model.  
  
With the V-model, the emphasis is on verification and validation. There is a relationship between the Development phase and the associated Test phase.  
  
The V-model reflects a management view and not a developer view, and is inflexible to changing requirements.  
  
The illustration shows the activities of the SDLC phases forming a V-shape. Coding is shown as the base of the V. The project definition phases form the left of the V. They start at the top with project & requirements planning, then product requirements and specification analysis, architecture high-level design, and detailed design, which is followed by coding.  
  
The test and integration phases form the right of the V and start after coding with unit testing, integration & testing, System & acceptance testing, and at the top is Production operation & maintenance.*   
  
The x-axis represents time, and the y-axis represents increasing levels of abstraction. The project cannot move back in time; once the phase is complete, it cannot be revisited. Each project definition phase is paired with a test and integration phase. Testing and integration is planned in parallel with its corresponding project definition phase. This ensures tests will exist to validate every deliverable. Being a sequential approach, the V-Model is easy to use and lets management track the project's progress with the milestones inherent in the SDLC. However, the V-Model still relies on a rigid structure, which reflects management's view rather than the dynamic nature of software projects. V-Model's inflexibility causes issues, when faced with concurrent events or dynamic changing requirements. Still, the V-Model's insistence on making QA activities a priority, makes it an excellent choice for systems requiring high reliability. This might include software used in medical devices, or hospital patient control applications.   
*The V is placed within a graph showing the movement in time as the X-axis and the level of abstraction as the Y-axis. The phase activities cause a constant increase in time, but the level of abstraction starts high at the top of the left of the V with project & requirements planning and gets less until it reaches Coding.   
  
The level of abstraction then increases with each phase as they move up the right side of the V-shape.  
  
The activities along the left and right of the V-shape that have corresponding positions are linked with an arrow: project and requirements planning is linked to production operation and maintenance, product requirements and specification analysis is linked to system and acceptance testing, architecture high-level design is linked to integration and testing, and detailed design is linked to unit testing.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

The Iterative Approach Software Method

Learning Objective

*After completing this topic, you should be able to*

* *use the iterative approach during software development*

**1. The iterative approach**

Adaptive software methods use a combination of iterative and incremental development, iterating to the Software Development Lifecycle multiple times during the project. First a small partial implementation of the total system is constructed, with little or no functionality. Then during each iteration through the SDLC, an incremental slice of functionality is added to the application and tested. Then the application's new features are demonstrated to the customer and their comments and observations are incorporated into the project's requirements influencing the features added in future iterations. The incremental model prioritizes the application's requirements and then implements the highest priority requirements first. At the end of each iteration, the application is left in a potentially deployable state, even if it has very few features that can be demonstrated. Each successive iteration adds features and functions to the software, leaving it in a state that can be deployed and demonstrated to the customer, until all required functionality has been implemented. Each iteration will include re-design and re-factoring, to accommodate changing requirements, streamline the existing code base, and enable the addition of new features.   
*Heading: Software Methods: The Iterative Approach.  
  
This is a combination of Iterative and Incremental methods. It includes multiple iterations of the SDLC in the project. The lessons learned now are used in future iterations. There is redesign and implementation at each iteration and the approach adapts to changing requirements.  
  
There are four iterative software models: Rapid Application Prototype, or RAD; Rational Unified Process, or RUD; Spiral; and Agile Software models.  
  
The illustration shows the approach as a series of concentric circles. The inner most circle consists of the Initial planning activities including Planning and Requirements. The next circle is slightly larger and the Analysis & design and Implementation activities are added. The next circle is slightly larger and the Testing and Evaluation activities are added.   
  
Although each new circle or iteration adds new activities, the process still iterates through the previous circles activities so that bugs and fixes can be implemented and the software is constantly being updated as it moves through the SDLC.  
  
The final circle is the Deployment phase.*   
  
Adaptive software methods include planning to identify milestones, but leave a lot of flexibility in how milestones are reached. And the plan is not rigid, the milestones can change. The focus is on adapting quickly to changing requirements and priorities. Common software models using the iterative approach include Rapid Application Prototyping, also known as RAD; Rational Unified Process, or RUP; the Spiral model; and the group of Agile software models. The iterative approach provides many benefits. Each iteration will deliver an operational product, which the customer could choose to deploy. And since requirements are prioritized, the highest risk or major functions will be delivered first. Customers have the opportunity to see the software demonstrated at the end of each iteration, giving feedback which guides future development. This ensures the product delivered at the end of the project, will meet the customer's expectations, even if those expectations change as the project is progressing. In fact, it is quite common for the customer's vision of the final product to evolve as they see new features demonstrated and get new ideas.   
*Heading: Iterative Approach Characteristics.  
  
The benefits of the iterative approach include each release delivers an operational product, the highest risk or priority features are implemented first, it allows for customer input after each iteration, and it is adaptive to changing requirements.  
  
Each iteration includes the Plan, Design, Testing, and Evaluation phase. This means that after each iteration the software is releasable.*   
  
These new ideas can be accommodated, because the iterative and incremental approach to software development adapts easily to changing requirements, reducing risk, and lowering the initial delivery cost. A project's success in using the iterative approach will be enhanced by using a senior development team with advanced skills in planning, design, and implementation. Delivering a slice of functionality in each iteration requires well-defined module interfaces, which must be planned out in advance, as they will be implemented in different iterations. And senior developers employ best practices to create well structured code, which facilitates the re-factoring needed in each iteration. The project must also have a well-defined goal, defining the complete and fully functional in system. This allows construction of a plan, for what will be included in each increment. Most of the requirements will be known upfront but are expected to evolve over time. Therefore, the development team will focus on near-term features guided by stated goals for the next release.   
*Heading: Iterative Approach Characteristics.  
  
The iterative approach requires engineers skilled in planning and design, as well as an early definition of the complete system.*   
  
A well-defined description of the In System also lets the project know, when it is done iterating, as it is clear when all the system's features have been delivered. The iterative approaches are well suited to the dynamic nature of software development. They excel, when a project has a lot of complexity and risk due to changing requirements or to using new, rapidly changing technology. The iterations give the development team time to learn the new technology in one cycle, delaying design decisions, until future cycles and adapting as the technology changes. Iterative approaches allow a project to identify risks and shifting expectations early and adapt as often as needed during the life of the project. Since software models using an iterative approach produce potentially deployable software after each iteration, they are also well-suited to projects where marketing demands require a quick release to get basic functionality to market early.   
*Heading: Iterative Approach Characteristics.  
  
The iterative approach is a good fit when you have lots of program risks and complexity such as evolving requirements and new technology, as well as marketing schedule demands.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

Agile Software Models

Learning Objective

*After completing this topic, you should be able to*

* *describe the values, practices and processes of the Agile Development Model*

**1. The Agile model**

Agile software models are lightweight, displaying iterative adaptive characteristics. They are less formal than sequential approaches, with reduced documentation and allow for rapid changes that are inherent in software development. Agile models are based on iterative and incremental development. A Software Development Lifecycle is iterated through multiple times, though adherence to the SDLC phases is not rigid. Different phases may be accelerated or bypassed. During each iteration, an incremental slice of functionality is added to the software and the resulting application is demonstrated to the customer. Customers give their feedback which is incorporated in the system’s requirements allowing them to evolve over the life of the project. The values and principles embodied in the Agile methods are well-defined. Leaders of the different Agile models met in 2001 to find common ground, which resulted in the Agile manifesto. The Agile manifesto describes the values demonstrated in all Agile software models. The Agile methods value individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan.   
*Heading: Agile Software Model Characteristics.  
  
Agile models are a group of iterative methods that are lightweight, iterative, and incremental, and with an emphasis on individuals and interactions, working software, customer collaboration, and responding to change.*   
  
Agile methods are characterized by an inspect and adapt mindset, which makes them highly flexible when faced with changing requirements. These methods excel in helping businesses respond to risky projects that are unpredictable. Agile methods are also well-suited to projects developing time-critical applications. Each iteration delivers and demonstrates a potentially shippable application, which can quickly be transformed into a full release. The different Agile software processes share some common practices. Many employ the test first concept of test-driven development. In this process a developer writes the unit tests to exercise a feature, before implementing the feature. The unit test is used to define the behavior of the feature. The unit test is run which fails because the feature has not yet been implemented. And then the developer implements the minimum amount of code necessary to provide the feature and pass the test. Once the unit test passes, the developer re-factors the code to meet approved coding standards. Test Driven Development encourages simple designs as only the code needed to pass the test is written.   
*Heading: Agile Software Model Characteristics.  
  
The Adaptive approach can adapt to changing requirements, responds to unpredictability, and deployment is possible at the end of any increment.  
  
It is illustrated as three circles, each circle represents the addition of another stage, and each one is larger than the previous one.   
  
The middle circle is the smallest and represented the iteration planning stage, the next stage adds implementation, and the largest circle adds delivery and feedback.  
  
Heading: Agile Software Practices and Models.  
  
Common Agile practices include TTD, or Test driven development, and user stories. The user story follows the formula of, "as who, I want what so that why?  
  
TTD is shown as circles. The middle circle write a failing test, the second circle makes the test pass, and the outside circle is the refractor.*   
  
This encourages the developer to focus only on what is important. The unit tests are often added to an automated test harness, which is run one or more times a day to ensure a modified code does not break existing functionality. Many Agile methods employ user stories to capture the features and functionalities the customer wants to see in the final product. The user story never describes how the feature will be implemented. Instead, it outlines who wants what feature and why in a simple statement, which gives the user's view of the system. A user story is meant to be open-ended and is often known as an invitation to a conversation with the customer in order to flesh out the details of the feature. User stories shift the focus from writing requirements to discussing them. The two best-known Agile processes are Extreme Programming, or XP, and Scrum. XP views the project through the eyes of the developer, focusing on implementation of the system, rather than management of the project. XP introduced and uses Test Driven Development. The customer is encouraged to write the unit test, which describe the behavior of the features.   
*Heading: Agile Software Practices and Models.  
  
XP (eXtreme Programming).  
  
In XP coding is a key activity and the emphasis is on simplicity.*   
  
The software is designed and implemented as simply as possible. XP projects are marked by heavy refactoring. Developers continuously restructure the code base to remove redundant code and complexity. Running the unit test ensures refactoring efforts do not change the system's behavior. The main activity throughout the XP software project is implementing the code. There is little documentation, and communication among teammates is done with the code. So coding standards are put in place so developers will implement all code according to the same rules. Pair programming is used, where two developers work together to code a module, one driving or typing, and the other observing. This leads to an attitude of collective ownership. Anyone can change any code anywhere in the system at any time. XP projects use continuous integration where the entire system is built and integrated every time a task is completed, sometimes several times in the same day.   
  
Scrum is the most popular Agile software method. In Scrum, the project is broken up into multiple development units called sprints. Each sprint is a time-boxed iteration of a complete SDLC. The sprint includes planning and analysis, design, development, test, and the delivery of potentially shippable software to the customer. At the end of each sprint, the product is demonstrated, and the customer gives feedback that can be incorporated in future sprints. Scrum contains three roles – the product donor, self-directed team, and the Scrum master. The product owner represents the customers and stakeholders and is responsible for capturing the project requirements and prioritizing them in the product backlog. The self-directed team is responsible for delivering potentially shippable software at the end of each sprint. And the Scrum master owns the Scrum process and ensures that it's followed. The Scrum master is also responsible for working with the team and removing impediments so the team can be as productive as possible.   
*Heading: Agile Software Practices and Models.  
  
Scrum.  
  
The Scrum approach consists of sprints, which are time-boxed iterations that are structured into five ceremonies. The approach allows for well defined roles.  
  
The approach is illustrated as a cycle starting with telling the story. This feeds into feedback from the stakeholder and managing the backlog, which feeds into planning a sprint.  
  
The sprint is performed and within the sprint are daily cycles. After the sprint is performed it is deployed to stakeholders, who are asked for feedback. This feeds into the Scrum.  
  
After the Scrum, the stakeholder gives feedback and the feedback is incorporated back into the user story.*   
  
Each sprint contains five meetings or ceremonies. In the backlog grooming meeting, the list of prioritized requirements is discussed and the top priority requirements are broken down into user stories, that describe features that can be developed in a single sprint. In the sprint planning meeting, the user stories that will be implemented in the next sprint are chosen and each is broken down into tasks describing how it will be accomplished. The daily Scrum is a short meeting where all team members check in and communicate their status. Each member of the team states what they did yesterday, what they will do today and any impediments blocking their way. In the sprint review, the potentially shippable software completed in the last sprint is demonstrated to the customer. And the sprint retrospective meeting is where the team reviews the last sprint, reflecting on the issues and what went well. Then the team puts actions in place to improve their performance in future sprints.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_wtfl_a01_it_enus/output/html/course_transcript.html#top)

© 2016 Skillsoft Ireland Limited