SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

7 Stages of the Software Development Life Cycle

SDLC MODEL

A framework that describes the activities performed at each stage of a software development project.

WATERFALL MODEL

Requirements – defines needed information, function, behavior, performance and interfaces.

Design – data structures, software architecture, interface representations, algorithmic details.

Implementation – source code, database, user documentation, testing.

Test - test the project on completion whether it is able to perform certain task properly.

Installation – install the application on the required system.

Maintenance – maintain the application in future.

WATERFALL STRENGTHS

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

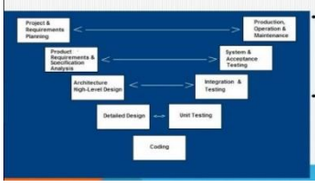
WATERFALL DEFICIENCIES

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

WHEN TO USE THE WATERFALL MODEL

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

V-SHAPED SDLC MODEL



- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-SHAPED STEPS

Project and Requirements Planning – allocate resources

Product Requirements and Specification Analysis – complete specification of the software system

Architecture or High-Level Design – defines how software functions fulfill the design

Detailed Design – develop algorithms for each architectural component

Production, operation and maintenance – provide for enhancement and corrections

System and acceptance testing – check the entire software system in its environment

Integration and Testing – check that modules interconnect correctly

Unit testing – check that each module acts as expected

Coding – transform algorithms into software

V-SHAPED STRENGTHS

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

V-SHAPED WEAKNESSES

- Does not easily handle concurrent events
- Does not handle iterations or phases
- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

WHEN TO USE THE V-SHAPED MODEL

- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- When it can be modified to handle changing requirements beyond analysis phase
- Solution and technology are known

RAPID APPLICATION MODEL (RAD)

RAPID APPLICATION MODEL (RAD)

- Requirements planning phase (a workshop utilizing structured discussion of business problems)
- User description phase – automated tools capture information from users
- Construction phase – productivity tools, such as code generators, screen generators, etc. inside a time-box. ("Do until done")
- Cutover phase -- installation of the system, user acceptance testing and user training

RAD STRENGTHS

- Reduced cycle time and improved productivity with fewer people means lower costs
- Time-box approach mitigates cost and schedule risk
- Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs
- Focus moves from documentation to code (WYSIWYG).
- Uses modeling concepts to capture information about business, data, and processes.

RAD WEAKNESSES

- Accelerated development process must give quick responses to the user
- Risk of never achieving closure
- Hard to use with legacy systems
- Requires a system that can be modularized
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

WHEN TO USE RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized

INCREMENTAL SDLC MODEL

Construct a partial implementation of a total system

Then slowly add increased functionality

The incremental model prioritizes requirements of the system and then implements them in groups.

Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.

INCREMENTAL MODEL STRENGTHS

- Develop high-risk or major functions first
- Each release delivers an operational product
- Customer can respond to each build
- Uses "divide and conquer" breakdown of tasks
- Lowers initial delivery cost
- Initial product delivery is faster
- Customers get important functionality early
- Risk of changing requirements is reduced

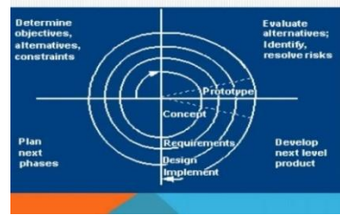
INCREMENTAL MODEL WEAKNESSES

- Requires good planning and design
- Requires early definition of a complete and fully functional system to allow for the definition of increments
- Well-defined module interfaces are required (some will be developed long before others)
- Total cost of the complete system is not lower

WHEN TO USE THE INCREMENTAL MODEL

- Risk, funding, schedule, program complexity, or need for early realization of benefits.
- Most of the requirements are known up-front but are expected to evolve over time
- A need to get basic functionality to the market early
- On projects which have lengthy development schedules
- On a project with new technology

SPIRAL SDLC MODEL



- Adds risk analysis, and 4gl RAD prototyping to the waterfall model
- Each cycle involves the same sequence of steps as the waterfall process model

SPIRAL MODEL STRENGTHS

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

SPIRAL MODEL WEAKNESSES

- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

WHEN TO USE SPIRAL MODEL

- When creation of a prototype is appropriate
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

Software Development Life Cycle

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

Steps in SDLC

- Communication – initiate the request
- Requirement Gathering – bringing out the information on what the project requirement is
- Feasibility Study - the team analyzes if a software can be made to fulfill all requirements of the user and if there is any possibility of software being no more useful.
- System Analysis - decide a roadmap of their plan and try to bring up the best

software model suitable for the project. Understanding the software limitations and changes to be done

- Software design - The inputs from users and information gathered in requirement gathering phase are the inputs of this step. Output will be the logical design and the physical design

- Coding – programming phase

- Testing – End user testing

- Integration – integration with other entity

- Implementation - installing the software on user machines

- Operation and Maintenance – maintaining the code, the systems for patch updates

System Implementation

Implementation

Implementation refers to activities that occur before the system is turned over to its users. Its purpose is to:

build a properly working system install it in the organization

- replace the old systems and work
- finalize system and user documentation
- train users
- prepare support systems to assist users

Coding

Coding is the process where the physical design specifications developed by the analysis team are converted into computer codes by the programming team. This can be an involved and intensive activity that depends on the size and complexity of the system.

Testing

To conduct a test, programmers should have already built the software and have in hand well-defined standards for what composes a defect. The product can be tested through reviewing their construction and composition or through exercising their function and examining the results. Software testing usually begins early in the systems development life cycle, although many of the actual testing activities are carried out during implementation.

Types of Testing

1. Unit testing
2. Integration testing
3. System testing

. Unit testing

- Unit Testing Sometimes called *module testing*, unit testing is the process of testing individual code modules before they are integrated with other modules. The objective of unit testing is to identify and fix as many errors as possible before modules are combined into larger software units (such as programs, classes, and subsystems). Unit testing is often automated but it can also be done manually.

Integration Testing

- Interface incompatibility. An example is a caller module that passes a variable of the wrong data type to a subordinate module.
- Parameter values. A module is passed or returns a value that was unexpected (such as negative number for a price).
- Run-time exceptions. A module generates an error such as “out of memory” or “file already in use” due to conflicting resource needs.

System Testing

- In system testing, instead of integrating modules into programs for testing, the programs are integrated into systems. Not only do individual modules and programs are tested several times in system testing, but also interfaces between modules and programs.

Acceptance Testing

This is testing the system in the environment where it will eventually be used. Acceptance means that users normally sign off on the system and “accept” it once they are satisfied with it.

There are two types of acceptance testing and these are:
1. Alpha testing – simulated data
2. Beta testing – real data in the real user environment

There are several types of tests done during alpha testing and these are as follows:

- Recovery testing – forces the software (or environment) to fail in order to verify that recovery is properly performed
- Security testing – verifies that protection mechanisms built into the system will protect it from improper penetration

- Stress testing – tries to break the system
- Performance testing – determines how the system performs in the range of possible environments in which it may be used.

Installation

Installation is the process of moving from the current information system to the new one. There are many constraints when installing a new system and making it operational.

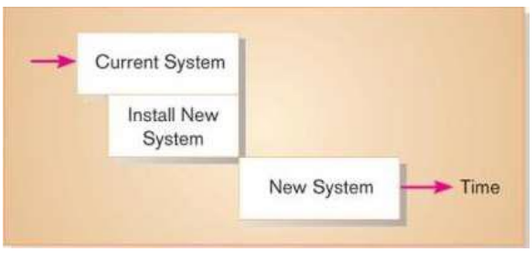
- Incurring costs operating both systems in parallel
- Detecting and correcting errors in the new system
- Potentially disrupting the company and its IS operations
- Training personnel and familiarizing customers with new procedures

Approaches to Installation

1. Direct installation
2. Parallel installation
3. Phased installation

1. Direct installation

In direct installation, the old system is turned off and the new system is turned on. In direct installation, the users use the new system. Whatever errors found from the new system will have a direct impact on the users and how they do their jobs. If the new system fails, considerable delay may occur until the old system can operate again and business transactions are re-entered to make the database updated.



Direct installation

Advantage:

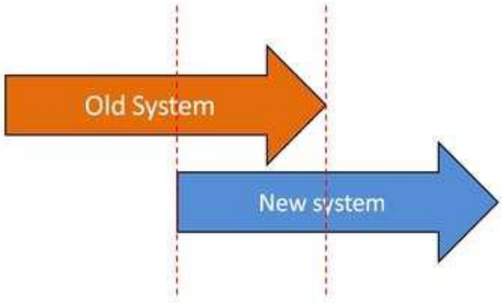
- Simple

Disadvantage:

- Risk

2. Parallel installation

In parallel installation, the old system continues to operate along with the new system until the users and management are satisfied with the performance of the new system and has been thoroughly tested and determined to be error-free and ready to operate independently.



2. Parallel installation

Advantage:

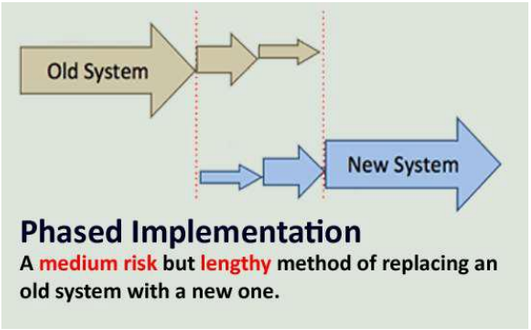
- Low risk of system failure

Disadvantage:

- Not Feasible

3. Phased installation

In Phased Installation, the new system is installed and brought into operation in a series of steps or phases. Components or functions are added to the operational system on each phase. The system is tested in each phase to make sure that it is ready for the next phase.



3. Phased installation

Advantage:

- Reduced risk

Disadvantage:

- Increased complexity

What is a Software

Software is capable of performing many tasks, as opposed to hardware which can only perform mechanical tasks that they are designed for. Software provides the means for accomplishing many different tasks with the same basic hardware

Classes of Software

System Software

Helps run the computer hardware and computer system itself. System software includes operating systems, device drivers, diagnostic tools and more. System software is almost always pre-installed on your computer.

Application Software

Allows users to accomplish one or more tasks. It includes word processing, web
browsing and almost any other task for
which you might install software.
(Some application software is preinstalled
on most computer systems.)

Programming Software

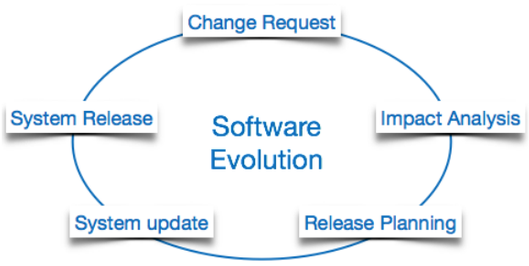
is a set of tools to aid developers in writing programs.
The various tools available are
compilers, linkers, debuggers,
interpreters and text editors.

Basic Principles

- Software, commonly known as programs or apps, consists of all the instructions that tell the hardware how to perform a task.
- These instructions come from a software developer in the form that will be accepted by the platform (operating system + CPU) that they are based on.
- For example, a program that is designed for the Windows operating system will only work for that specific operating system. Compatibility of software will vary as the design of the software and the operating system differ.
- Software, in its most general sense, is a set of instructions or programs instructing a computer to do specific tasks.

Software Evolution

The process of developing a software product using software engineering principles and methods is referred to as software evolution. This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.



Laws of Software Evolution

Continuing Change

A software system must continue to adapt to the real world changes, else it becomes progressively less useful.

Increasing Complexity

A software system evolves, its complexity tends to increase unless work is done to maintain or reduce it.

Conservation of Familiarity

The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc. must be retained at any cost, to implement the changes in the system.

Continuing Growth

In order for a system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business.

Reducing Quality

A software system declines in quality unless rigorously maintained and adapted to a changing operational environment.

Feedback Systems

The software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.

Self-regulations

A system evolution processes are self-regulating with the distribution of product and process measures close to normal.

Organizational Stability

The average effective global activity rate in an evolving a system is invariant over the lifetime of the product.
Software Paradigms

Software Development Paradigm

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand.

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of the following

Requirement gathering
• Software design
• Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes the following:

Design
• Maintenance
• Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes
Coding
• Testing
• Integration

Characteristics of good software

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

Operational
• Transitional
• Maintenance

Operational

- Budget - cost
- Usability - the degree of ease with which products such as software and Web applications can be used to achieve required goals effectively and efficiently.
- Efficiency - is defined as a level of performance that uses the lowest amount of inputs to create the greatest amount of outputs.
- Correctness - adherence to the specifications that determine how users can interact with the software and how the software should behave when it is used correctly.
- Functionality - is the ability of the system to do the work for which it was intended.
- Dependability - is the ability to provide services that can defensibly be trusted within a time-period.
- Security – secured system
- Safety

Traditional

This aspect is important when the software is moved from one platform to another:

- Portability - he usability of the same software in different environments
- Reusability - the use of existing assets in some form within the software product development process; these assets are products and by-products of the software development life cycle and include code, software components, test suites, designs and documentation.

This aspect is important when the software is moved from one platform to another:

- Interoperability - the ability of computer systems or software to exchange and make use of information
- Adaptability - an open system that is able to fit its behavior according to changes in its environment or in parts of the system itself

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the everchanging environment:

- Modularity the extent to which a software/Web application may be divided into smaller modules.
- Maintainability - is defined as the degree to which an application is understood, repaired, or enhanced.
- Flexibility - it normally refers to the ability for the solution to adapt to possible or future changes in its requirements.

This aspect briefs about how well a software has the capabilities to maintain itself in the everchanging environment:

- Scalability - is the ability of a program to scale

Data Gathering Guidelines

- Focus on identifying the stakeholders
- Involve all the stakeholder groups
- Need more than in person from stakeholder group(s)
- Use a combination of data gathering techniques
- Support the data-gathering sessions with suitable props, such as task descriptions and prototypes if available.
- Run a pilot session if possible, to ensure that your data-gathering session is likely to go as planned

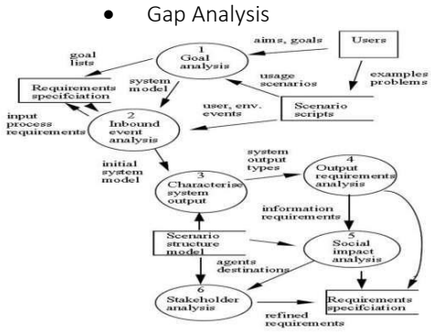
Requirements Analysis

Requirements Analysis is the process of defining the expectations of the users for an application that is to be built or modified. Requirements analysis involves all the tasks that are conducted to identify the needs of different stakeholders.

Requirements Analysis Techniques

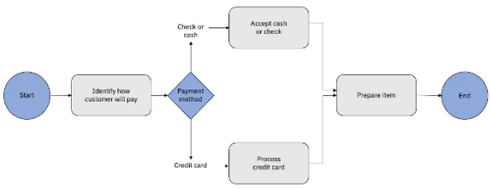
- Business process modeling notation (BPMN)
- Flowchart technique
- Data flow diagram
- Role Activity Diagrams (RAD)

- Gantt Charts
- Gap Analysis



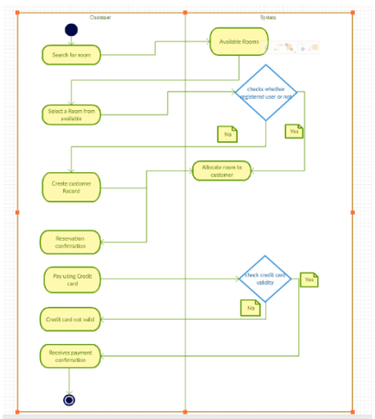
Business Process Modeling Notation (BPMN)

Business process modeling and notation is used to create graphs for the business process. These graphs simplify understanding the business process. BPMN is widely popular as a process improvement methodology.



Flowchart technique

A flowchart depicts the sequential flow and control logic of a set of activities that are related. Flowcharts are in different formats such as linear, cross-functional, and top-down. The flowchart can represent system interactions, data flows, etc.

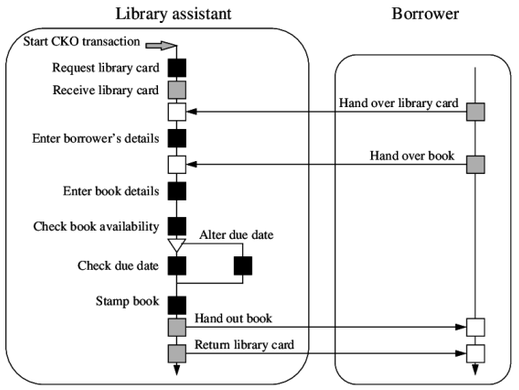


Data flow diagram

Data Flow Diagram describes various entities and their relationships with the help of standardized notations and symbols. By visualizing all the elements of the system it is easier to identify any shortcomings. These shortcomings are then eliminated in a bid to create a robust solution.

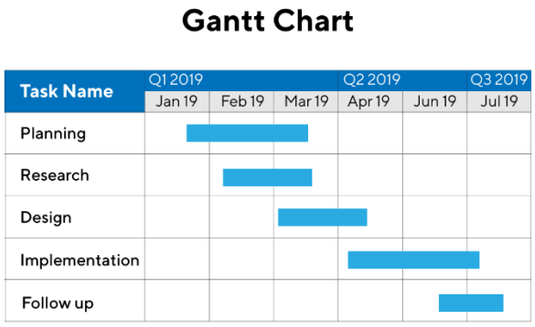
Role Activity Diagrams (RAD)

Role-activity diagram (RAD) is a role-oriented process model that represents role activity diagrams. Role activity diagrams are a high-level view that captures the dynamics and role structure of an organization.



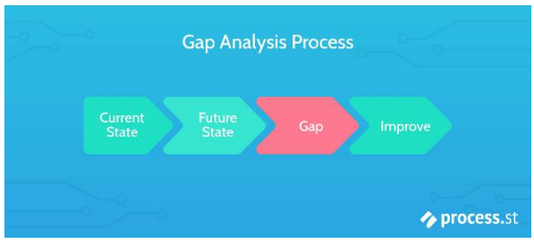
Gantt Charts

Gant charts used in project planning as they provide a visual representation of tasks that are scheduled along with the timelines. The Gant charts help to know what is scheduled to be completed by which date. The start and end dates of all the tasks in the project can be seen in a single view.



Gap Analysis

Gap analysis is a technique which helps to analyze the gaps
in performance of a software application to determine whether the business requirements are met or not. It also involves the steps that are to be taken to ensure that all the business requirements are met successfully.



Requirements
Gathering

Requirements Gathering is a fundamental part of any software development project. It is the process of generating a list of requirements (functional, system, technical, etc.) from all the stakeholders (customers, users, vendors, IT staff) that will be used
as the basis for the formal definition of what the project is.

The Importance of Requirements Gathering

- The problem with the business customers is that the latter tends to expect software teams to deliver a solution based on unspoken, incomplete or unknown requirements.

- The problem with software teams is that they tend to assume that business customers will communicate exactly what they want as succinctly as possible.

- The requirements need to be formally captured in one document that can be used as a reference during software development.

The Importance of Requirements Gathering

- Good gathering, processing and management of requirements is important as it sets clear targets for everyone to aim for. It can be a lot of hard work, but it need not be a daunting task if you can keep some key points in mind.

- Consider always the user inputs. Find out how actual users complete their tasks and exactly how they get things done – and how they want to get things done.

The Users Comes First

- The requirements should detail how a user would accomplish what they want using the software being developed. Awareness of any technological preferences and existing system integration is also fundamental, as it can have a huge impact on the development path and subsequently impact on performance and user task efficiency.

What are requirements and its specifications?

- A requirement is a statement about an intended product that specifies what it should do or how it should perform.

- Goal:
  - To make as specific, unambiguous, and clear as possible.

- Functional Specifications:
  - What the system should do

- Non-Functional Specifications:
  - what constraints there are on the system its development - (For example that a work processor runs on different platforms)

What requirements should be gathered?

- Functional:
  - What the product should do.
- Data requirements:
  - Capture the type, volatility, size/amount, persistence, accuracy and the amounts of the required data.
- Environmental requirements:
  - a) context of use
  - b) Social environment (eg. Collaboration and coordination)
  - c) how good is user support likely to be d) what technologies will it run on

- User Requirements:
  - Capture the characteristics of the intended user group.
- Usability Requirement:
  - Usability goals associated measures for a particular product.

Data Gathering Techniques

- Questionnaires: Series of questions designed to elicit specific information from us.
  - The questions may require different kinds of answers: some require a simple Yes/No, others ask us to choose from a set of pre-supplied answers.

- Interviews: Interviews involve asking someone a set of questions.
  - Often interviews are face-to-face
  - Forum for talking to people
  - Structured, unstructured or semi-structured
  - Props, e.g. sample scenarios of use, prototypes, can be used in interviews
  - Good for exploring issues
  - But are time consuming and may being feasible to visit everyone

- Focus groups and workshops
  - Interviews tend to be one on one, and elicit only one person's perspective. It can be very revealing to get a group of stakeholders together to discuss issues and requirements.

- Studying documentation:
  - Procedures and rules are often written down in a manual and these are a good source of data about the steps involved in an

activity and any regulations governing a task.

- Naturalistic Observation
    - It can be very difficult for humans to explain what they do or to even describe accurately how they achieve a task.

    - Spend time with stakeholders in their day-to-day tasks, observing work as it happens

    - Gain insights into stakeholders' tasks

    - Good for understanding the nature and context of the tasks

    - But it requires time and commitment from a member of the design team, and it can result in a huge amount of data

    - Ethnography is one form: entire class devoted to this.