The software development life cycle (SDLC) is the process of planning, writing, modifying, and maintaining software.

The systems development life cycle originally consisted of five stages instead of seven. These included planning, creating, developing, testing, and deploying. Note that it left out the major stages of analysis and maintenance.

The new seven phases of SDLC include planning, analysis, design, development, testing, implementation, and maintenance.

**7 Stages of the System Development Life Cycle**

There are seven primary stages of the modern system development life cycle. Here’s a brief breakdown:

* Planning Stage
* Feasibility or Requirements of Analysis Stage
* Design and Prototyping Stage
* Software Development Stage
* Software Testing Stage
* Implementation and Integration
* Operations and Maintenance Stage

### **Phase 1: Planning or Requirement collection**

The first stage in the SDLC process is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry.

Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage.

**Description**:

Preliminary step to understand and document project goals and user needs.

**Objective:**

Understand and document the needs of stakeholders to form a solid foundation for software development.

* **Gathering requirements**: Understand what the customer and stakeholders need.
* **Feasibility study**: Evaluating technical, economic, and operational viability.
* **Documenting**: Creating a Requirement Specification Document.

**Note**: Inadequate requirement analysis is a leading cause of project failure.

**1. Requirement Gathering**

**Description**: Engage with stakeholders to understand and collect system needs.

* + **Techniques**:
    - **Interviews**: One-on-one discussions with stakeholders.
    - **Questionnaires**: Structured forms distributed to stakeholders.
    - **Observations**: Watching end-users in their natural environment to understand tasks and challenges.
    - **Workshops**: Collaborative sessions with key stakeholders.
    - **Brainstorming**: Group activities to generate ideas and requirements.
    - **Prototyping**: Creating mock-ups or basic versions of the software to gather feedback.

**Note**: Diverse stakeholder engagement ensures comprehensive requirement coverage.

**2. Feasibility Study**

**Objective**

Determine the viability of the project before significant resources are committed to it.

**Description**: Evaluate the practicality of the project in various aspects.

* + **Types**:
    - **Technical Feasibility**: Can we build it with our current technology and expertise?
    - **Economic Feasibility**: Is the project financially viable? Cost-benefit analysis is used here.
    - **Operational Feasibility**: Will the system work in the real-world operational scenario of the organization?
    - **Legal Feasibility**: Are there any legal concerns or constraints related to the system?
    - **Time Feasibility**: Can the project be completed within the necessary time frame?

**Note**: A project might be feasible in one area but not in others. It's crucial to consider all aspects.

**a). Technical Feasibility**

**Description**: Assesses whether the technical resources and expertise available are adequate for the project.

**Considerations**:

* + - Do we have the necessary technology to build the system?
    - Do the project's technical requirements align with the current infrastructure?
    - Is the required technology mature or still emerging?
    - Are there gaps in technical knowledge that need to be addressed?

**Note**: This helps in recognizing whether it's technically possible to convert the idea into a fully functional software.

**b). Economic (or Financial) Feasibility**

**Description**: Evaluates whether the project is financially viable.

**Considerations**:

* + - What is the estimated cost of the project?
    - What are the expected financial benefits once the software is deployed?
    - Are there potential financial risks, and if so, how can they be mitigated?
    - Is the Return on Investment (ROI) acceptable?

**Note**: It's crucial to have a clear financial blueprint to prevent overshooting budgets and ensuring a good return.

**c). Operational Feasibility**

**Description**: Determines if the system will function and be accepted in the intended real-world scenario.

**Considerations**:

* + - Will the proposed system solve the current operational challenges?
    - How will the end-users receive the new system? Will they adapt to it?
    - What changes in the operational process does the new system require?

**Note**: A technically perfect system might still fail if it's not operationally practical or accepted by its users.

**d). Legal Feasibility**

**Description**: Assesses if there are any legal implications or constraints related to the project.

**Considerations**:

* + - Are there any data privacy laws that the software needs to comply with?
    - Are there intellectual property concerns?
    - What are the legal obligations related to the software's domain or functionality?

**Note**: In today's environment, with stringent data protection laws like GDPR, this aspect is especially vital for software handling user data.

**e). Time Feasibility**

**Description**: Evaluates if the project can be completed within the required time frame.

**Considerations**:

* + - What's the estimated time to complete the project?
    - Are there any hard deadlines or time constraints?
    - How does the project timeline align with other organizational activities or market events?

**Note**: Time overruns can lead to increased costs, missed market opportunities, and stakeholder dissatisfaction.

In essence, the feasibility study is a risk assessment mechanism. It gives stakeholders the confidence to proceed, knowing that major potential challenges have been considered and addressed. While the feasibility study offers valuable insights, it's worth noting that some level of uncertainty always remains. The goal is to minimize this uncertainty as much as possible.Top of Form

**3. Requirement Documentation**

**Objective :** Capture and describe the system's requirements, ensuring clarity, traceability, and agreement among all stakeholders.

**Description**: Document all the gathered and analyzed requirements in a structured manner.

**Artifacts**:

* + **Functional Requirement Specification (FRS)**: Describes functionalities the software should provide.
  + **System Requirement Specification (SRS)**: Comprehensive description of system behavior, including functional and non-functional requirements. Often serves as a contract between stakeholders and developers.
  + **Use Cases**: Represents interactions between the system and its users, defining potential scenarios and system behavior.

**A). Functional Requirement Specification (FRS)**

**Description**: This document describes functionalities the software should provide. It provides detailed specifications of all functionalities that the system needs to perform.

**Components**:

* + - **Introduction**: Brief overview and objectives of the document.
    - **Functional Requirements**: Detailed description of each feature the software should offer.
    - **Use Cases**: A description of how users will interact with the software.
    - **Input/Output Requirements**: Description of the input data the system will receive and the output data it should produce.
    - **Data Handling and Storage**: Requirements related to data processing, storage, and retrieval.

**Note**: The FRS ensures that functionalities are clearly defined, leaving no ambiguity in what the system should accomplish.

***Functional Requirement Specification (FRS) for Online Reservation System***

***Document Version: 1.0***

***Date: [Insert Date]***

***Author: [Insert Author Name]***

***1. Introduction***

*This document lays out the functional requirements for the Online Reservation System, a feature of our broader accommodation booking platform. It is intended to define the system's expectations from a functionality viewpoint.*

***1.1 Objective***

*To establish a clear and shared understanding of the system's functionalities among stakeholders, developers, and testers.*

**B). System Requirement Specification (SRS)**

**Description**: A comprehensive description of the intended purpose and environment of the software, including both functional and non-functional requirements.

**Components**:

* + - **Introduction**: Overview, scope, and objectives.
    - **Overall Description**: A broader view of the system, its functionalities, and its interactions.
    - **Specific Requirements**: This can encompass both the functional and non-functional requirements. Examples include performance requirements, design constraints, software system attributes (like reliability, security), etc.
    - **Appendices**: Any relevant reference documents, glossaries, or indices.

**Note**: The SRS is often treated as a contract between the client and the developers. Any changes to this document usually require formal agreement from both parties.

**C). Use Cases**

**Description**: Represents interactions between the software (or a system) and its users, defining potential scenarios and the system's behavior.

**Components**:

* + - **Actor**: Any entity that interacts with the system (e.g., user, external system).
    - **Scenario**: Step-by-step process showing how the actor interacts with the system.
    - **Preconditions**: Conditions that must be true before a use case starts.
    - **Postconditions**: Conditions that are true after the use case has been executed.
    - **Extensions**: Variations of the scenario, often detailing exceptions or errors.

**Note**: Use cases are particularly valuable in understanding user interactions and can often be the basis for testing scenarios later in the SDLC.

**D). Requirement Traceability Matrix (RTM)**

**Description**: A document that maps and traces user requirements with test cases. It ensures each requirement is fulfilled by the system.

**Components**:

* + - **Requirement ID**: A unique identifier for each requirement.
    - **Requirement Description**: A brief description of the requirement.
    - **Source of Requirement**: Where the requirement originated from.
    - **Test Cases**: List of test cases that ensure the requirement is satisfied.

**Note**: The RTM is vital for ensuring no requirement is overlooked throughout the development and testing phases.

Requirement documentation is a continuous process. As the project progresses, changes might be made, requiring the documentation to be updated accordingly.

**Note**: Requirement documentation should be clear, concise, and devoid of ambiguities. It's often revisited throughout the SDLC.

**4. Requirement Validation**

* **Description**: Ensure that the requirements are correct, complete, unambiguous, and feasible.
  + **Activities**:
    - **Review Sessions**: Meetings where team members, stakeholders, and sometimes even domain experts review the requirements.
    - **Prototyping**: Using prototypes to validate requirements with stakeholders.

**Note**: Addressing errors or gaps in this phase is more cost-effective than during later stages of the SDLC.

**5. Requirement Management**

* **Description**: As the project progresses, requirements might change or evolve. Managing these changes is essential.
  + **Activities**:
    - **Change Control**: A structured process to accept, document, and incorporate changes to the requirements.
    - **Traceability**: Ensuring each requirement is traceable back to its source and forward to its implementation and delivery.

**Note**: Requirement changes are inevitable in most projects, especially long ones. Effective management ensures the project remains on track.Top of Form

Remember, the requirement analysis phase's quality directly impacts the success of the subsequent phases. Proper time and resources should be invested to ensure its thoroughness and accuracy.

Requirements Gathering stage needs teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

### **Phase 2: Design**

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

**High-Level Design (HLD)**

* Brief description and name of each module
* An outline about the functionality of every module
* Interface relationship and dependencies between modules
* Database tables identified along with their key elements.
* Complete architecture diagrams along with technology details

**Low-Level Design (LLD)**

* Functional logic of the modules
* Database tables, which include type and size.
* Complete detail of the interface
* Addresses all types of dependency issues.
* Listing of error messages
* Complete input and outputs for every module

### **Phase 3: Coding**

Once the system design phase is over, the next phase is coding. In this phase, developers start building the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

### **Phase 4: Testing**

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer's requirements.

During this phase, the QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and sends it back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

### **Phase 5: Installation/Deployment**

Once the software testing phase is over and no bugs or errors are left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

### **Phase 6: Maintenance**

Once the system is deployed, and customers start using the developed system, the following 3 activities occur.

* Bug fixing – bugs are reported because of some scenarios that are not tested at all.
* Upgrade – Upgrading the application to the newer versions of the Software.
* Enhancement – Adding some new features to the existing software.

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specifications mentioned in the first phase.

## Popular SDLC Models

Here, are some of the most important models of Software Development Life Cycle (SDLC):

### [Waterfall model in SDLC](https://www.guru99.com/what-is-sdlc-or-waterfall-model.html)

The waterfall is a widely accepted SDLC model. In this approach, the whole process of the software development is divided into various phases of SDLC. In this SDLC model, the outcome of one phase acts as the input for the next phase.

This SDLC model is documentation-intensive, with earlier phases documenting what need be performed in the subsequent phases.

### [Incremental Model in SDLC](https://www.guru99.com/what-is-incremental-model-in-sdlc-advantages-disadvantages.html)

The incremental model is not a separate model. It is essentially a series of waterfall cycles. The requirements are divided into groups at the start of the project. For each group, the SDLC model is followed to develop software. The SDLC life cycle process is repeated, with each release adding more functionality until all requirements are met. In this method, every cycle act as the maintenance phase for the previous software release. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.

### [V-Model in SDLC](https://www.guru99.com/v-model-software-testing.html)

In this type of SDLC model testing and the development, the phase is planned in parallel. So, there are verification phases of SDLC on the side and the validation phase on the other side. V-Model joins by Coding phase.

### [Agile Model in SDLC](https://www.guru99.com/agile-scrum-extreme-testing.html)

Agile methodology is a practice which promotes continue interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks.

### [Spiral Model](https://www.guru99.com/what-is-spiral-model-when-to-use-advantages-disadvantages.html)

The spiral model is a risk-driven process model. This SDLC testing model helps the team to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc.

This model adopts the best features of the prototyping model and the waterfall model. The spiral methodology is a combination of rapid prototyping and concurrency in design and development activities.

### **Big bang model**

Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come.

This model works best for small projects with smaller size development teams that are working together. It is also useful for academic software development projects. It is an ideal model where requirements are either unknown or the final release date is not given.

----------------------------------------------------------------------------------------------------------------

X Paths:

----------------------------------------------------------------------------------------------------------------

A diagram of an employee

Description automatically generated with medium confidence

Xpath=//tagname[@attribute='value']

Basic Format of XPath

* **// :** Select current node.
* **Tagname:**Tagname of the particular node.
* **@:** Select attribute.
* **Attribute:** Attribute name of the node.
* **Value:** Value of the attribute.

XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

|  |  |
| --- | --- |
| **Expression** | **Description** |
| *nodename* | Select all nodes with the name "*nodename*" |
| / | Selects from the root node |
| // | Selects nodes in the document from the current node that match the selection no matter where they are |
| . | Selects the current node |
| .. | Selects the parent of the current node |
| @ | Selects attributes |

|  |  |
| --- | --- |
| **Path Expression** | **Result** |
| **bookstore** | **Select all nodes with the name "bookstore"** |
| **/bookstore** | **Select the root element bookstore**  **Note: If the path starts with a slash ( / ) it always represents an absolute path to an element!** |
| **bookstore/book** | **Selects all book elements that are children of the bookstore** |
| **//book** | **Selects all book elements no matter where they are in the document** |
| **bookstore//book** | **Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element** |
| **//@lang** | **Selects all attributes that are named lang** |

**Types of X-path**

There are two types of XPath:

**1) Absolute XPath**

**2) Relative XPath**

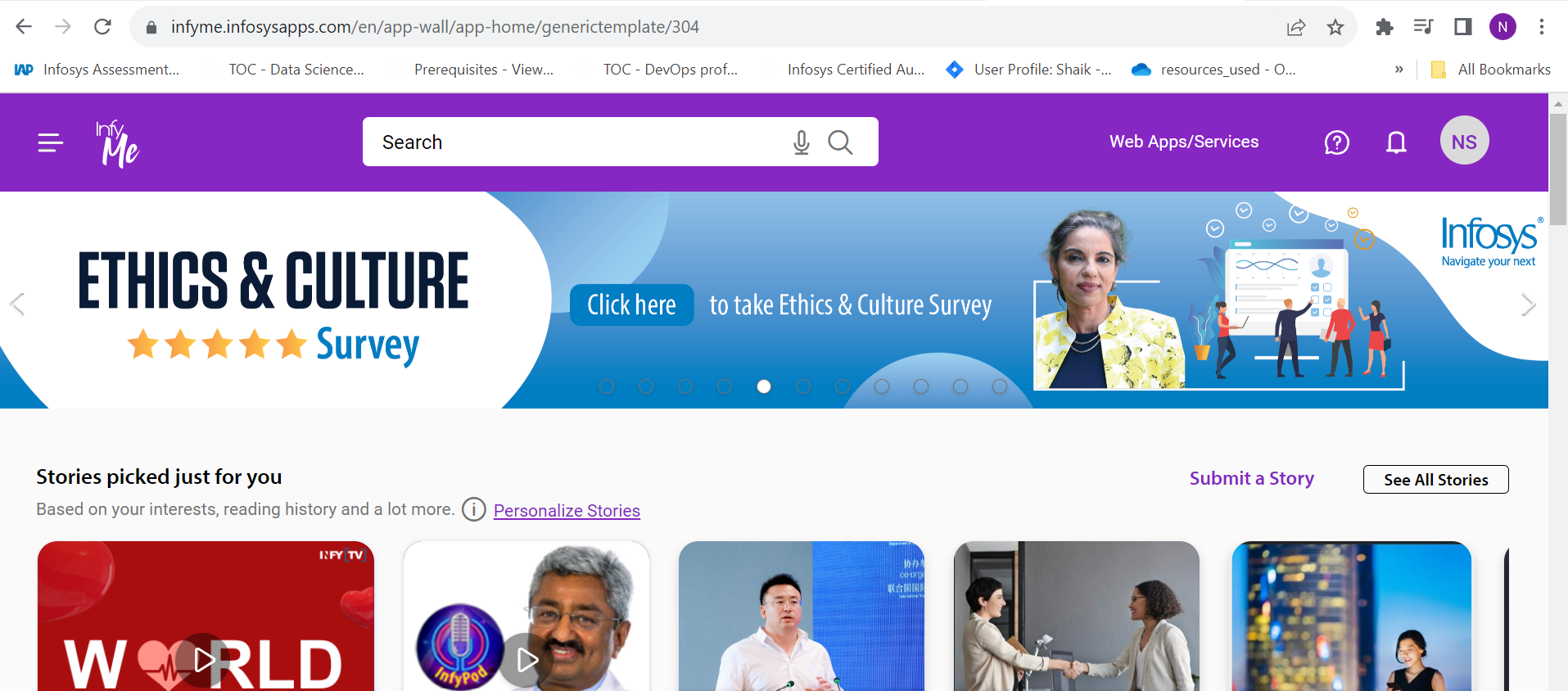
**Absolute XPath:**

It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath fails.

The key characteristic of XPath is that it begins with the single forward slash(/), which means you can select the element from the root node.

Let's take a website Infyme web of having the URL as follows:

<https://infyme.infosysapps.com/en/app-wall/app-home/generictemplate/304>



Absolute Xpath of the web element “***Web Apps/services***” is as follows:

//\*[@id="mainHeader"]/ul/li[1]/div/button/span

