Digital Government Authority
هـيـئـة الحكومة الرقمـيـة

Guideline for Software Development Life Cycle (SDLC) Methodology

June 2024

Document Type: **Guideline**
Document Classification: **Public**
Issue No: **1.0**
Document No: **DGA-1-2-5-229**

Contents

1. Introduction

The Digital Government Authority (DGA) realizes the importance of continuously approving and updating organizations to keep pace with current and future requirements, and to mainly contribute to enhancing digital performance within government agencies, increasing the quality of services provided and improving the experience of the beneficiaries of such services, in line with Saudi Arabia's ambitious Vision 2030 and the strategic directions of the digital government, which emphasize the importance of establishing an effective and flexible regulatory environment that adapts to future changes. By paving the way for government agencies to offer high-quality and efficient digital government services, DGA aims to increase investment returns and boost the national economy.

DGA's Guideline for Software Development Life Cycle (SDLC) Methodology aims to support and empower government agencies by providing guidance on adopting best practices related to digital government, and to achieve goals of the Kingdom's Vision 2030 as well as the strategic goals of government digital transformation. By following this Guideline, agencies can adopt digital methodologies to overcome challenges, improve performance, enhance quality of digital services, increase user satisfaction, and achieve digital excellence and sustainability. The Document provides an overview of SDLC methodology, including models, tools, and steps for developing and operating software.

2. Guideline Objectives

This Guideline aims to achieve the following:

- Supporting government agencies by providing guidance on the steps of applying Software Development Life Cycle (SDLC) methodology.
- Promoting adoption of SDLC methodology.
- Improving performance of government agencies in systems and software development field.
- Enhancing quality of digital government services and products.

3. Guideline Scope

This Guideline aims to facilitate the understanding of SDLC methodology and its adoption process by focusing on the following aspects:

- Software Development Life Cycle (SDLC) methodology;
- Objectives of SDLC methodology;
- Stages of implementing SDLC;
- Models supporting the application of SDLC methodology;
- General recommendations for adopting SDLC methodology;
- Continuity and impact measurement.

4. Target Audience

This Guideline is intended for experts, practitioners, and technical staff involved in system and software development within government agencies.

5. Guideline Statement

5.1 SDLC Methodology

Software Development Life Cycle (SDLC) methodology is an approach to software development that encompasses various processes and steps and addresses numerous concepts, such as requirements analysis, design, development, testing, deployment, and operation. SDLC methodology's key outcomes include promoting effective collaboration among software developers in building technical solutions that meet business needs. This, in turn, enhances the quality level and reduces costs.

5.2 Objectives of SDLC Methodology

Software Development Life Cycle (SDLC) methodology is a comprehensive and valuable approach. By applying this methodology, five main objectives can be achieved:

Faster development

Implementing the methodology helps reduce the time required for system and application development.

Ensuring successful software development

Following the methodology ensures the success of the development process by systematically understanding and analyzing the purpose and requirements of the technical solution.

Cost reduction

Properly utilizing the methodology increases efficiency and reduces development costs.

Higher quality

Adhering to the methodology during development enhances the quality of the outputs.

Supporting effective collaboration

This methodology promotes collaboration among software developers by providing a structured process for finding suitable technical solutions that meet business needs.

5.3 Stages of Applying SDLC Methodology

Software Development Life Cycle (SDLC) methodology consists of six fundamental stages, as illustrated in Fig. 1:

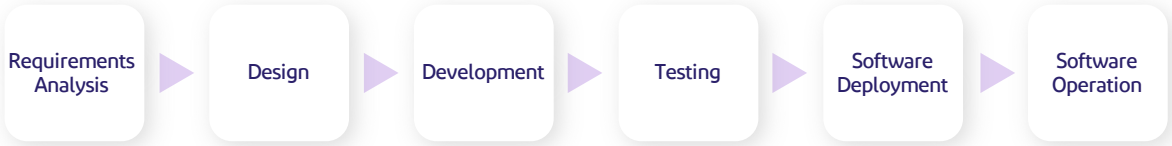

Fig. 1: Stages of Applying SDLC Methodology

Requirements analysis

This stage involves studying current needs, planning new system or program, and determining requirements for its implementation.

Design

This stage involves designing proposed system or program, and engineering technical solutions.

Development

This stage involves building the new system or program, and transforming visualized design into computer software (CSCI).

Testing

This stage involves conducting comprehensive testing to ensure that all aspects of the new system or program align with the initial vision.

Software Deployment

This stage involves deploying the system or program in the production environment to begin its usage.

Software Operation

This stage involves carrying out activities such as maintenance, monitoring, and improvement, to ensure continuity.

The following provides a detailed explanation of each stage involved in the application of SDLC methodology:

5.3.1 Requirements Analysis

Requirements analysis is the process of describing the behavior, characteristics, and features of the software to be developed:

Business Requirements (BR)

These clarify business objectives (key broader outcomes towards which efforts and actions should be directed), business objectives (measurable steps to achieve objectives), business roles and departmental needs, (requirements for activities carried out by organization), and business processes (necessary activities and tasks to accomplish organization goals).

User Requirements (UR)

These define user needs and expected activities within the technical solution, including user experience (that drives user interaction), accessibility (addresses distinct aspects), and usability (design effectiveness) aspects through capturing user needs.

Functional Requirements (FR)

These describe the capabilities and expected operational benefits of the technical solution, including:

- **Features and User Stories**: Requirements written from the perspective of an end user describing their needs.
- **Use cases**: Requirements to interact with the technical solution to meet the end user needs.
- **Generic Requirements (Epic)**: General business requirement that can be divided into project requirements.

Technical Requirements (TR)
Technical aspects of the solution to be adopted include reliability, scalability, security, performance, integration, interoperability, standards, architectures, interfaces, deployment, transportation, environment, safety, and human factors. These requirements cover all technical constraints related to the construction, design, deployment, and operation of the technical solution.

System Requirements
This focuses on the aspects related to systems and applications.

There are four main activities in the above-outlined requirements analysis process:

User Needs Analysis
This involves estimating the extent to which specific user needs can be met using current software and hardware technologies. This analysis shall be conducted through various studies, including impact analysis on implementing these requirements and their effect on current systems.

Requirements Elicitation
Requirements are gathered through the observation of existing systems, discussions with potential users, requirements workshops, creation of storyboards, etc.

Requirements Specification
Collected information shall then be converted into "Requirements Documents" that include business requirements, user requirements, functional requirements, and technical requirements.

Requirements Verification
This activity ensures the completeness and accuracy of the generated documents based on user needs. A feasibility study of implementing the requirements can also be conducted.

5.3.2 Design

The design stage involves visualizing the technical solution based on the collected and analyzed requirements. The technical solution is composed of components arranged at different levels (Fig. 2), including the following:

User Experience Level

This level manages interaction with end users through browsers or applications, utilizing a set of services provided at the service level.

Service Level

This level includes appropriate architecture patterns and defines communication mechanisms to achieve operational benefits.

Business and Workflow Layer Level

At this level, operational features are managed.

Data and Analysis Level

This level involves analysis and utilization of data resulting from operations.

Infrastructure Level

This level encompasses all hardware, networking, and non-software tools required for the technical solution.

Supportive Levels

Additional supporting levels, such as integration level, knowledge level, and security level, add value and support the development of a complete, robust, and suitable technical solution.

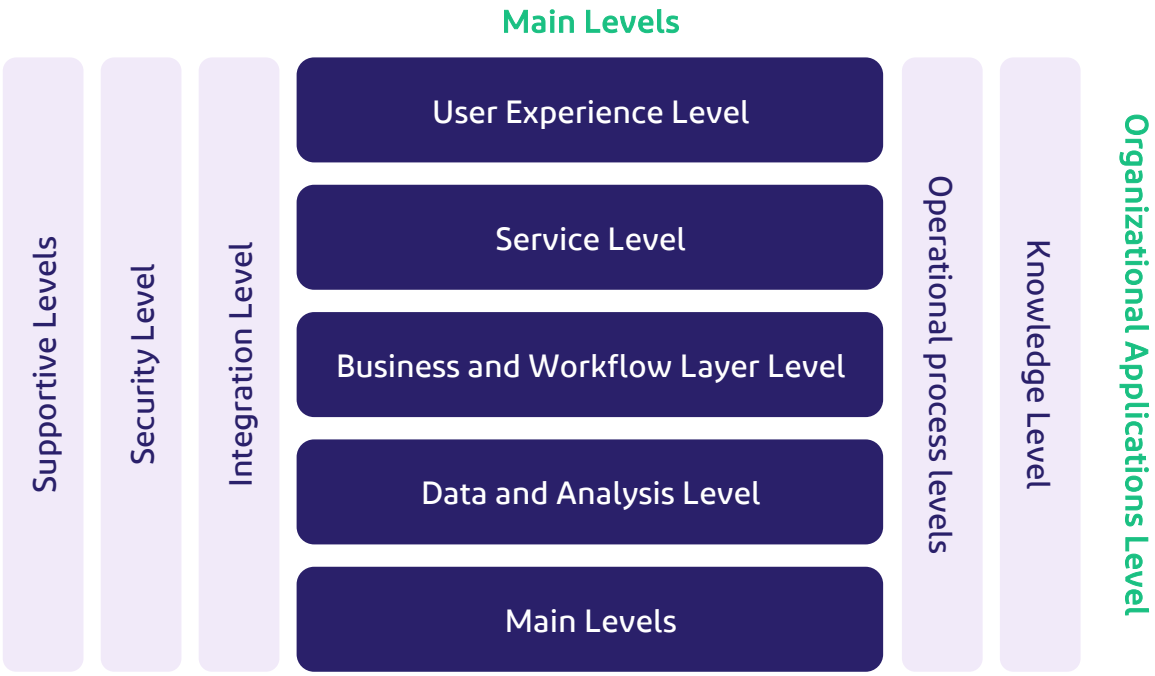

Fig. 2: Technical Solution Components and Levels

The design processes include the following steps:

Technical Solution Engineering

This focuses on analyzing software into layers, basic units, and subunits, highlighting their behavior and interactions.

Technical Solution Design

The emphasis is on solving problems by utilizing design solutions that are reusable, maintainable, flexible, robust, and widely applicable to address common design challenges.

The design process encompasses various activities to ensure consistent and comprehensive engineering and design of the technical solution.
Below is an outline of some of the main activities:

User Experience and Interface Design

This covers all aspects of end-user interaction with the technical solution and involves making decisions regarding interactions, navigation, and accessibility of services based on user categories and authorities.

Engineering Design

This determines decisions related to software structure, layers, modules, and how interfaces are designed between modules.

Component Design

This involves customizing services and exposing interfaces to other components.

Data Structure Design

This focuses on designing data models and structures.

Algorithmic Design

This encompasses decisions related to algorithms required to fulfill the services provided by each component or part of the system.

Software Design

This repetitive, multi-cycle process typically consists of at least three cycles that produce a number of designs as follows:

High-Level Design (HLD)

HLD focuses on the nature of components and their interactions based on the technical solution's architecture and design.

Low-Level Design (LLD)

LLD is an initial refinement of overall design, providing detailed definitions of the actual logic of each component. It addresses build tools, programming languages, frameworks, libraries, products, data structures, as well as performance and security algorithms.

Detailed Design (DLD)

DLD is a further stage of design refinement, addressing aspects like error handling, algorithm implementation decisions, logical sequences, and data structure optimizations.

5.3.3 Development

Software development is the process of transforming a software design into a functional, reliable, and integrated Computer Software Configuration Item (CSCI) that is ready for software testing. The development process involves collaboration between software developers and graphic designers to create a CSCI that aligns with the specified requirements. The development process includes the following steps:

- Enhancing the detailed design to accommodate algorithmic and data structure implementation decisions.
- Engaging in the creative process to design graphics that meet user requirements (e.g., UX, UI) and align with software design choices.
- Writing source code, compilers, configuration files, and software resources, and utilizing source code management tools to facilitate sharing and collaboration among developers.
- Writing automation code to expedite the development process or enable continuous development practices.
- Writing script code to interact with operating systems, databases, servers, and other components of the technical solution.
- Identifying and utilizing compatible libraries, frameworks, tools, and products aligning the design decisions.
- Designing unit tests to ensure that the code meets requirements and design choices.
- Generating technical documentation using code-based methods.
- Conducting unit testing of the built tools through unit test suites and cases, utilizing unit testing frameworks.
- Integrating and assembling software components into larger components, applications, products, and solutions.
- Designing and performing automated integration tests to qualify modules of logically integrated software and assess the impact on individual module behavior.
- Grouping tools into deployable, composable, and/or executable packages that can be shared as CSCIs.
- Creating prototypes and testing software components to address implementation complexities and risks, or to create a proof of concept (POC).

- Resolving issues and bugs in existing software, creating new versions or submission packages for deployment in the operating environment.
- Defining and managing versions, including file versions, module versions, and generation sequences for compilers.
- Tracking source code modifications and releases using SDLC tools, such as to-do lists, issue tracking lists and tools, and managing functional feature additions.

5.3.4 Testing

Software testing is a crucial process that verifies if the software developed in the previous stage meets the expected requirements and is free from programming errors. Testing is conducted in dedicated environments separate from the main production environment, yet identical thereto, and should be an integral part of the comprehensive quality assurance and control process to ensure compliance with requirements. The testing process encompasses the following key activities:

Test Planning and Monitoring

This involves creating a document that outlines the overall approach and objectives of the testing process.

Test Analysis and Construction of Test Steps

This involves conducting further analyses to the design through the following steps:

- Reviewing test basis, which includes requirements, design specifications, product risk analysis, architecture, and interfaces.
- Determining necessary test conditions.
- Creating test environment and identifying required infrastructure and tools.
- Creating test cases.

Test Execution

This entails performing the specified test either manually or using automated testing tools. The activities include:

- Executing designed, updated, and prioritized test cases.
- Re-executing previously failed tests.
- Recording the results of test execution, including previously failed tests.
- Comparing actual results with expected ones.

Test Monitoring
This involves comparing actual progress with test plan and generating status reports, including cases of deviations from the plan.

Defining Exit Criteria
This includes determining when to stop testing, preparing test reports, and evaluating the need for further testing.

Test Completion and Closure Criteria
Establishing criteria for test completion and assessing software's readiness for implementation.

The testing process encompasses both functional and non-functional tests, illustrated as follows:

Functional Testing
This testing relates to functional requirements, and focuses on qualifying the built software modules and Computer Software Configuration Items (CSCI) against functional and user requirements.

Non-functional Testing
This testing qualifies the built software modules and CSCI against technical requirements.

Functional and non-functional tests can be performed manually, automatically, or using a combination of both as follows:

Manual Testing
It involves using software as an end user would, following specific steps in test cases and test suites to compare expected results with actual ones and identify potentially recurring failures or issues.

Automated Testing
This involves writing scripts or configuring testing tools against CSCI, simulate user or application interactions in predefined and pre-designed test cases.

Semi-automated Testing
This combines automated tools with manual testing techniques.

5.3.5 Software Deployment

Software Deployment is the process of preparing an application or system to run and operate in a specific environment. It involves installation, configuration, to ensure optimal software performance, integration, migration, update, disablement, uninstallation, version tracking, and running multiple versions of the same software simultaneously. This includes ensuring its compatibility with the network infrastructure and hardware.

At the beginning of the deployment process, a copy of the production environment, called "pre-production," is prepared. It undergoes deployment verification and new release processes for the product before actual application in the production environment. In case of deployment failure, a "rollback" activity is performed, which involves re-deploying the last working version of the product.

There are several types of software deployment:

On-premise Deployment

It refers to internal deployments within the entity, with full control over software management and integration with other components. On-premise deployments allow both online and offline access, and special considerations for networks and security are important in this type of deployment.

Cloud Deployment

It refers to external deployments with full control over software management and limited control over integration with other components, depending on the service type. The software or system is accessed online, considering the hosting provider's requirements. The hosting provider handles network, security, and accessibility considerations. Cloud deployment can be further divided into various types:

- Infrastructure as a Service (IaaS)
- Function as a Service (FaaS)
- Software as a Service (SaaS)
- Data as a Service (DaaS)
- Platform as a Service (PaaS)

Cloud deployment offers more flexibility, adaptability, security, manageability, and scalability.

Virtualized Deployment
The process of deploying software in virtualized environments independently of the underlying physical infrastructure.

Containerized Deployment
The process of deploying applications, products, and solutions along with their software dependencies, including networks, operating system (OS) configurations, and disk partitions, to run the software in a completely isolated environment (container).

5.3.6 Software Operation
Software operations encompass a range of activities aimed at enhancing operational efficiencies of software, increasing productivity, and reducing operational costs by overcoming operational hurdles and automating repetitive tasks, which involve intricate details. Among the common operational activities are:

Solution Operations
This represents the primary role of the operational process, and includes: enhancing the performance and usability of the technical solution.

Operational environment management
This encompasses infrastructure, OS, configuration, networks, and storage. Operational environment management can be categorized into several types, as outlined below:

- **Infrastructure management**: This involves setting up and configuring the required infrastructure for deployments.
- **Version management**: This includes managing the assignment of the version and configuration properties for the software to be deployed.
- **Network management**: This covers any network-related configurations, such as virtual IPs for high accessibility.
- **Storage management**: This includes configuring and managing storage-related services.
- **Directory services management**: This entails partitioning stored data for identity, users, and software components to support access control operations
- **User support**: This includes direct engagement and interfacing with end-users to identify usage problems and software bugs related to infrastructure configuration or that require intervention in the development process. This interaction is tracked through the help desk.

Disaster mitigation

The process of removing or minimizing the impact of risks and their sources by taking preemptive measures, such as performing database and configuration backups, before and after a disaster or emergency event, and taking actions to restore the latest version from the backup to enable the smooth operation of the technical solution.

Solution monitoring

This encompasses continuous monitoring to anticipate any software operation failures, including:

- **Performance monitoring**: This involves tracking resource usage and the software's response time to verify if it aligns with the expected behavior. It also includes interpreting resource usage trends to anticipate necessary actions to avoid any service disruptions
- **Log monitoring**: This entails monitoring detailed output logs of the solution or application to detect any abnormal events indicating recurring or unhandled bugs.
- **Data monitoring**: This includes monitoring the volume of data used by the solution to plan for potential disk expansions.
- **Security and threat monitoring**: This includes leveraging security mechanisms, systems, and tools to address any security threats.

Reporting, alerting, and warning

When the solution, application, or its environment exhibits unexpected behavior, appropriate actions must be taken by the relevant stakeholders (developers, security personnel, testers).

5.4 Supporting Models for System Development Life Cycle (SDLC) Methodology Application

Supporting models for SDLC methodology application are growing. Choosing the right model can make a significant difference in achieving successful results in terms of cost, on-time delivery, customer satisfaction, or the safety and quality of the software code.

Supporting model types for SDLC methodology application can be classified into two main categories (Figure 3), as follows:

Sequential Models

Models that divide software development activities into sequential and linear steps, where each step of the activity depends on the results of the previous activity. The most common methodologies for the sequential model are the waterfall methodology and the Sashimi model.

Iterative Models

In iterative models, the activities are arranged sequentially, including activities of analyzing requirements, developing design, deployment, and operational processes to produce a recurring cycle. The cycle is repeated in this model until the work is completed. Iterative models can be further divided into the following two models:

- **Spiral Models**: Recurring cycles that focus on risk management and improve components based on those risks.
- **Incremental Models**: Incremental development of software, where the completion and verification process is carried out for each added component, and then it is integrated into the software as a whole before moving to the next cycle. One of the common methodologies for the incremental model is Agile, which relies on small increments within short time periods (a few weeks) in iterative cycles.

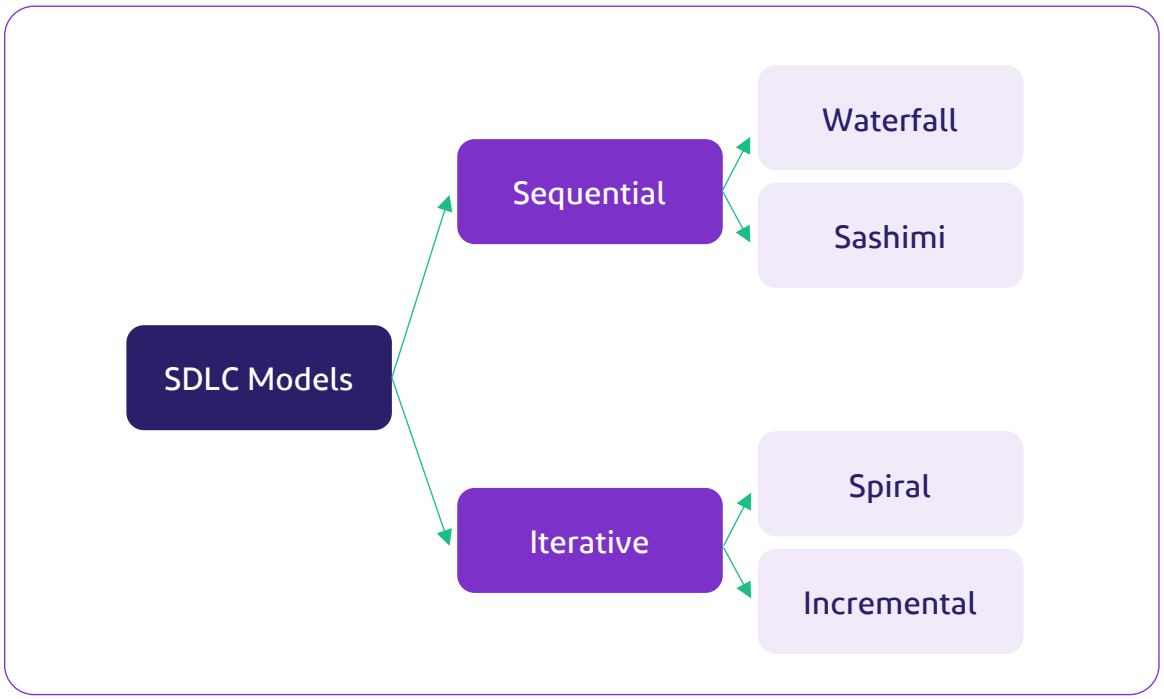

Figure (3) Models used in SDLC methodology

The suitable model is chosen based on the characteristics and constraints of each project. Various elements and use cases are considered when choosing methodologies affiliated with models that are suitable for the work, based on the following:

Waterfall methodology is used in the following cases:

- When the requirements are well-defined and unlikely to change significantly.
- When all the details (or most of them) can be collected before commencing work.
- In cases where the work team is proficient in handling the technology used.
- When SOW and its schedule are limited, allowing control over any deviations resulting from requirement enhancements or changes. This ensures containment within acceptable cost boundaries.

Agile methodology is used in the following cases:

- When stakeholders are capable and willing to participate in the work iteratively.
- In scenarios where the primary requirements are defined but details are not entirely clear.
- When requirements are subject to change during the project execution period.
- If consideration is given to exploring new technology or operational environments not yet mastered by the team.
- In cases where the team consists of a small number of individuals.
- When there is a need to showcase software workflow to stakeholders on a frequent basis.
- When the priority in the work focuses more on adherence to completion timelines than being output and feature-centric.
- In cases where work items are small and easily estimable, such as simple issues or minor improvement requests.

5.5 General Recommendations for SDLC Methodology Adoption

The following checklist can serve as a comprehensive reference for the necessary steps to ensure the successful application of SDLC methodology and the identification of key requirements:

- Identify all work requirements.
- Identify all user and stakeholder requirements.
- Identify all functional requirements.
- Identify all technical requirements.
- Identify all security requirements.
- Obtain approval and endorsement from stakeholders for all requirements.
- Identify software and hardware requirements, providing detailed insights for development, testing, deployment, and monitoring phases.
- Identify all acceptance criteria.
- Subject the project schedule, budget, and software requirements specifications to thorough reviews by the project team, the sponsoring entity, and relevant stakeholders.
- Adopt and align estimates and plans with the overall business strategy.
- Document and disseminate the design (high-level design, detailed design).
- Verify code compliance with organizational rules and standards.
- Conduct unit tests and integration tests for any source code.
- Maintain versions for code, documentation, and any information related to the utilized code.
- Ensure the performance of all necessary tests, encompassing automated user interface testing, load and stress testing, and any other relevant automated tests.
- Manual testing is limited to scenarios where automation is impractical.
- Enable system monitoring and application monitoring for all components and core resources..

5.6 Continuity and Impact Measurement

- To ensure the proper adoption of SDLC methodology, specific key performance indicators (KPIs) can be measured before, during, and after the adoption phase. These indicators can be categorized into three main types: financial, customer-related, and internal performance indicators, aiming to achieve the following impacts:
- Speed to market: Measure the time spent on system development before and after the adoption of methodologies, and continually track progress.
- Greater Success: Success is gauged through customer satisfaction, as adhering to organized methodologies in system development contributes to satisfying beneficiaries and users of these systems.
- Lower Costs: Evaluate the costs associated with system development before and after adopting methodologies to ensure the realization of intended savings.
- Higher Quality: Assess the final quality outcomes of systems post-application of methodologies.

The following are some KPIs that government entities can assess and monitor to gauge the efficacy of methodology application. It should be noted that the percentages are subject to modification based on the scale of the entities and projects

KPI	Description	Top-Tier Performance	Average Performance	Poor Performance
Satisfaction rate	Percentage of end-user satisfaction with the system	More than 90%	70 – 90%	Less than 70%
Percentage of change	Number of change requests made to SOW compared to the base SOW	Less than 10%	10 – 20 %	More than 20%
Failure rate	Percentage of system bugs	0-15%	15-30%	More than 30%
Support ticket number	Number of support tickets raised within a specific time frame after the system's utilization.	Less than 10%	10 – 20 %	More than 20%

6. Definitions Table

Unless the context otherwise requires, the following expressions and terms, wherever mentioned herein, shall have the meanings ascribed thereto.

Term	Definition
DGA	Digital Government Authority
Digital Transformation	Transforming and strategically developing business models into digitally-enabled models based on data, technology, and communication networks.
Digital Government	Supporting administrative, organizational, and operational processes within and across government sectors to achieve digital transformation and to develop, enhance, and facilitate easy and effective access to government information and services.
Government Entities	Ministries, authorities, public institutions, councils, and national centers, and their equivalents.
System Development Life Cycle (SDLC)	A methodology for developing systems and software that consists of multiple sequential stages.
Computer Software Configuration Item (CSCI)	A set of software or any of its separate components that meet and implement the functional requirements requested and specified by stakeholders
Modules	One of the components of software or a part of a software that contains one or more procedures. One or more independently developed units can form the entire software, and each unit serves unique and separate operations.
Infrastructure Patterns	Description of the software structure and design of the types of connections between its components.
Agile	A software development process through which software is created and developed through the collaborative efforts of a specific multi-functional work team.
Iterative Models	Types of SDLC models and divide the development process according to distinct activities for software development sequentially and linearly. Each activity's tool relies on the outputs of the previous activity.
Waterfall Model	A project management model that focuses on linear progress from the beginning to the end of the project. This methodology focuses more on the early stages, such as detailed planning, detailed documentation, and sequential execution.
Sashimi Model	One of SDLC models in which the sequential stages overlap, where a stage starts before the completion of the previous stage, unlike the waterfall model, to allow for review and correction of any problems or deficiencies that arise when starting a new stage.
Business Requirements	It illustrates business goals, objectives, roles, department needs, and business processes.
User Requirements	Requirements that clarify user needs and activities expected from the system user within the solution, including UX that defines user interaction, accessibility, usability, inclusion, goals, and constraints.
Test Cases	A description of a potential scenario that occurs on the component or system, with an explanation of the inputs for each test case and the expected outputs or results.
Production Environment	The space where the latest version of the software is published, which is in working condition, free of errors, and available when the user needs it.
Staging environments	A nearly identical replica of the production environment used for software testing. Staging environments are designed to test codes, configurations, and updates to ensure quality in an environment similar to production before software deploying.
Software Deployment	The process of preparing a software application to run and operate in a specific environment, including installation and configuration to ensure optimal software operation.
Storyboards	A tool used in agile business analysis to create visual models of user stories and help identify potential problems and risks. Storyboarding is used to describe a task, scenario, or story in terms of how stakeholders interact with the solution.
Containers	Virtual simulation at the level of OS applications, across multiple resources in networks, allowing software applications to run in isolated user spaces called "containers" in any cloud or non-cloud environment.

7. Abbreviation Table

Abbreviation	Meaning
SDLC	Software Development Life Cycle
BR	Business Requirements
CSCI	Computer Software Configuration Item
DaaS	Data as a Service
DLD	Detailed Level Design
FaaS	Function as a Service
FR	Functional Requirements
HLD	High Level Design
IaaS	Infrastructure as a Service
LDAP	Lightweight Directory Access Protocol
LLD	Low Level Design
PaaS	Platform as a Service
POC	Proof of Concept
SaaS	Software as a Service
TR	Technical Requirements
UR	User Requirements

هيئة الحكومة الرقمية
Digital Government Authority

DGA.GOV.SA