# Plan de Aseguramiento de Calidad de Software

# Documentación de Versionamiento

## Lista de Revisiones

**\*A** – Adición; **M –** Modificación; **D** – Eliminación

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Versión** | **Fecha** | **Número de la Figura, Párrafo,**  **Documento completo** | **A\* M D** | **Título o breve descripción** | **Número de Solicitud de Cambio** |
|  |  |  |  |  |  |
| 1.0 | 28/07/2025 | Sección 5 | **A** | Creación de archivo de roles y modificación de plantilla de herramientas | Solicitud N° 1 |
|  |  |  |  |  |  |
| 1.1 | 28/07/2025 | Encabezado de archivos | M | Se modificó el encabezado de 2 archivos | Solicitud N° 2 |
| 1.2 | 29/07/2025 | Archivo de roles | M | Se modificó el archivo de roles a la nueva plantilla | Solicitud N° 3 |

## Documento de Solicitud de Cambio

|  |  |
| --- | --- |
| Título del Documento: **Plan de Aseguramiento de Calidad de Software** | Número: |
| Organización que hace la solicitud: | |
| Contacto: | Tel: |
| Correo: | |
| Título: | Fecha: |
| Ubicación: | |
| Cambio propuesto: | |
| Justificación para el cambio: | |

**Sección Página**

[Plan de Aseguramiento de Calidad de Software 1](#_Toc204719596)

[Documentación de Versionamiento 1](#_Toc204719597)

[Lista de Revisiones 1](#_Toc204719598)

[Documento de Solicitud de Cambio 2](#_Toc204719599)

[Sección 1 – Objetivo 4](#_Toc204719600)

[1.1 – Alcance 4](#_Toc204719601)

[1.2 – Panorámica del Sistema 4](#_Toc204719602)

[1.4 – Relación con otros Planes 4](#_Toc204719603)

[Sección 2 – SQA Management 5](#_Toc204719604)

[2.1 – Estructura Organizacional SQ 5](#_Toc204719605)

[2.2 – Estándares, prácticas y convenciones 7](#_Toc204719606)

[2.3 – Esfuerzo, recursos y calendarios 8](#_Toc204719607)

[2.4 – Herramientas 9](#_Toc204719608)

[Sección 3 – SQA Tarea 20](#_Toc204719609)

[3.1 – Tarea: Proceso de Evaluación de Análisis de Requisitos de Software 20](#_Toc204719610)

[3.2 – Tarea: Proceso de Evaluación del Diseño 22](#_Toc204719611)

[3.3 – Tarea: Evaluación del Proceso de Implementación 23](#_Toc204719612)

[Herramientas utilizadas para la evaluación de implementación 24](#_Toc204719613)

[3.4 – Tarea: Evaluación del Proceso de Pruebas 26](#_Toc204719614)

[3.5 – Tarea: Evaluación del Proceso de Gestión de Despliegue y Liberación 27](#_Toc204719615)

[Sección 4 – Documentación 28](#_Toc204719616)

[**4.1 – Documento de Requisitos de Software** 28](#_Toc204719617)

[**4.2 – Informes de Pruebas de Software** 29](#_Toc204719618)

[**4.3 – Arquitectura y Diseño de Software** 30](#_Toc204719619)

[**4.4 – Documentación de Usuario** 32](#_Toc204719620)

[Sección 5 – Medición de la Calidad del Software 34](#_Toc204719621)

[Sección 6 – Capacitación 35](#_Toc204719622)

[Sección 7 – Informe y Resolución de Problemas de SQA 36](#_Toc204719623)

[7.1 – Incidencias del Proceso de Calidad 36](#_Toc204719624)

# Sección 1 – Objetivo

El objetivo de este plan es definir el Aseguramiento de la Calidad del Software (SQA) para el proyecto EduGestion. Se establecen las tareas y responsabilidades, se incluyen referencias a documentos y guías para la ejecución de las actividades de SQA, y se proporcionan los estándares, prácticas y convenciones a utilizar, así como las herramientas, técnicas y metodologías necesarias para llevar a cabo y documentar dichas actividades.

## 1.1 – Alcance

El plan establece las actividades SQA a lo largo del ciclo de vida de desarrollo del proyecto.

Este plan asegura que el software y su documentación cumple con los requisitos funcionales, de negocio, y técnicos esperados.

**Tabla 1. Actividades del Ciclo de Vida del Software**

|  |  |
| --- | --- |
| **FASE CVDS** | **ACTIVIDAD** |
| DEFINICIÓN | Planificación del Proyecto |
| Entorno de Desarrollo de Software |
| Análisis de Requisitos |
| DISEÑO | Diseño del Sistema |
| Análisis de Requisitos |
| Diseño del Software |
| DESARROLLO | Pruebas Unitarias |
| PRUEBA | Pruebas de Integración y Regresión |
| Pruebas de Verificación y Validación |
| DESPLIEGUE/RELEASE | Preparación del Software para su Uso |
| Preparación de Transición |
| MANTENIMIENTO | Mantenimiento a lo largo del Ciclo de Vida |

## 1.2 – Panorámica del Sistema

EduGestion es una plataforma web para la gestión académica integral. Permite registrar estudiantes, docentes y cursos, gestionar calificaciones, y generar reportes académicos. El sistema está diseñado con arquitectura MVC, estilo cliente-servidor y utiliza servicios RESTful. No presenta dependencias críticas con otros sistemas externos más allá del servidor institucional donde será desplegado.

**1.3 – Definiciones y Acrónimos**

• SQA: Software Quality Assurance  
• CVDS: Ciclo de Vida del Desarrollo de Software  
• RF: Requisito Funcional  
• RNF: Requisito No Funcional  
• SCM: Software Configuration Management  
• CRUD: Crear, Leer, Actualizar, Eliminar  
• MVC: Modelo Vista Controlador

## 1.4 – Relación con otros Planes

Este documento hace referencia directa a los siguientes planes y entregables del proyecto:

- Plan de Desarrollo de Software (versión 1.0 – 2025-06-28)

- Plan de Implementación (versión 1.0)

- Plan de Software Configuration Management

- Documento de Requisitos de Software (SRS)

- Documento de Diseño de Software

- Matriz de Trazabilidad de Requisitos

- Minutas de Reuniones del proyecto

# Sección 2 – SQA Management

## 2.1 – Estructura Organizacional SQ

El equipo SQA interactúa con el Tester QA, desarrolladores y analistas para la revisión de entregables. El jefe de Proyecto delega funciones y tiene autoridad para liberar productos. Los conflictos se escalan al jefe de Proyecto y se resuelven con base en las minutas.

**Figura 1. Estructura organizacional del proyecto EduGestion**

Diagrama

El contenido generado por IA puede ser incorrecto.

**Tabla 2. Equipo de SQA (Roles y Responsabilidades)**

|  |  |  |
| --- | --- | --- |
| **Rol** | **Asignado** | **Responsabilidades** |
| Coordinador de Calidad | Fernando Lezcano | * Asegura que las actividades de calidad se planifiquen, coordinen y ejecuten en todos los grupos. * Supervisa la aplicación general del plan SQA. |
| Líder de Evaluación de Requisitos y Gestión | Edgar Lorenzo | * Dirige las tareas de revisión del análisis de requisitos, diseño, implementación, pruebas y despliegue. * Coordina tareas de auditoría de procesos técnicos. |
| Líder de Documentación de Calidad | José Bustamante | * Supervisa la revisión, trazabilidad y validación de los documentos clave: requisitos, pruebas, arquitectura, usuario. * Verifica cumplimiento con listas de verificación SQA. |
| Líder de Métricas y Medición de Calidad | Linette Bonilla | * Define, recopila y evalúa métricas de calidad para cada etapa del SDLC. * Apoya en análisis de defectos y rendimiento (Sección 5). |
| Revisor(es) de Calidad (SQA Reviewers) | Emanuel González  Edwin Hou  Diego Corrales  Jesse Kazim  Eriol Tuñón  Cristopher Quintero  María Ferrer  Miguel Rodríguez | * Realizan revisiones cruzadas de entregables de otros grupos. * Identifican defectos y aseguran cumplimiento con normas y estándares. |
| Analistas SQA | Edward Camaño  Gonzalo Hooker  María Madrid  Josué pino  Daniel Pérez  Miguel Arosemena  Angel Márquez  Anabely Abrego  Lisbeth Magallon  Felix Beitia  Eloy Tulipano  María Ramírez  Alberto Davis | * Apoyan actividades de seguimiento, auditoría y documentación de calidad. * Realizan trazabilidad entre requisitos, pruebas, métricas y entregables. |
| Especialista Técnico (Arquitectura / Diseño) | Melanie Martinez  Rafaela Candanedo  Luz De León | * Da soporte técnico en revisión de diseño y arquitectura del sistema. * Apoya validación de documentación técnica. |
| Redactor Técnico / Documentador | Danna Dawkins  Abigail Koo  María Quiñones | * Encargado de estandarizar y asegurar la claridad en documentación de usuario y técnica. |
| Arquitectos de Software | Imanol Rodríguez  Nayim Rodríguez | * Aseguran que la arquitectura sea escalable, segura y mantenible. * Verifican consistencia entre requisitos y diseño técnico. |
| Evaluadora de Calidad de Implementación | Paola Ran | * Reporta métricas de calidad de código: densidad de defectos, cumplimiento de estándares y buenas prácticas de codificación. |
| Coordinadora de Revisiones | Tiphany Díaz | * Organiza y registra resultados de inspecciones. Mide efectividad de revisiones mediante número de hallazgos detectados. |
| Evaluador de Resultados de Pruebas | Steven Ampie | * Mide cobertura de pruebas, porcentaje de casos exitosos y severidad de defectos. Asegura integridad de resultados. |
| Tester | Hernán Solano | * Verifica que el software cumpla con los requisitos funcionales. Documenta y reporta cualquier defecto encontrado. |
| Auditor de Calidad de Software | Diego Osorio | * Evalúa el cumplimiento normativo, revisa políticas y procedimientos. Emite informes de hallazgos sin participar en codificación o pruebas. |
| Especialista en Métricas de Calidad | Emiola Fagbemi | * Responsable de métricas clave como DDE (Defect Detection Efficiency), DRE (Defect Removal Efficiency) y MTTR (Mean Time To Repair). |

**Tabla 3. Matriz RACI**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rol** | **Asignado(a)** | **Definir** | **Diseñar** | **Desarrollar** | **Probar** | **Desplegar** |
| Lider Coordinador de Calidad | Fernando Lezcano | A | C | C | C | I |
| Líder de Evaluación de Requisitos y Gestión | Edgar Lorenzo | C | C | C | R | C |
| Líder de Documentación de Calidad | José Bustamante | I | C | R | I | I |
| Líder de Métricas y Medición de Calidad | Linette Bonilla | C | R | I | C | I |

## 2.2 – Estándares, prácticas y convenciones

|  |  |  |  |
| --- | --- | --- | --- |
| **Categoría** | **Nombre o Descripción** | **Aplicación en el Proyecto** | **Referencia / Fuente** |
| Estándar de calidad | IEEE 730-1998 | Guía principal para estructurar el plan SQA | IEEE |
| Gestión de versiones | Git y GitHub | Control de código fuente y ramas | SCM |
| Nomenclatura | v1.0, v1.1… | Identificación de versiones | SCM |
| Diseño arquitectónico | MVC | Organización lógica del sistema | Diseño |
| Gestión de requisitos | Matriz de Trazabilidad | Relación entre casos de uso y pruebos | Trazabilidad |
| Requisitos | IEEE SRS | Especificación de RF y RNF | SRS |

## 2.3 – Esfuerzo, recursos y calendarios

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fase del CVDS** | **Rol responsable** | **Frecuencia** | **Horas estimadas** | **Mes / Sprint** |
| **Requisitos** | Analista | Semanal | 16h | Semana 1-2 |
| **Diseño** | Desarrolador Líder | Semanal | 16h | Semana 3-4 |
| **Implementación** | Desarrolladores | Diario | 80h | Semana 5-8 |
| **Pruebas** | Tester QA | Diario/Semanal | 32h | Semana 9-10 |
| |  | | --- | | **Despliegue** |  |  | | --- | |  | | Jefe Proyecto | Una vez | 8h | Semana 11 |
| |  | | --- | | **Seguimiento y reportes** |  |  | | --- | |  | | Jefe Proyecto |  | 4h | Todas |
| **Total estimado** |  | | 156h |  |

## 2.4 – Herramientas

**1. Herramientas para organización y entrega de los entregables**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Criterio | Peso | ClickUp (0-10) | Po | Trello (0-10) | Po | Miro (0-10) | Po |
| Usabilidad + | 5 | 9 | 45 | 9 | 45 | 9 | 45 |
| Personalización+ | 4 | 10 | 40 | 6 | 24 | 10 | 40 |
| Funcionalidades+ | 5 | 10 | 50 | 7 | 35 | 9 | 45 |
| Costo- | 4 | -3 | -12 | -4 | -16 | -2 | -8 |
| Integración y Compatibilidad + | 4 | 8 | 32 | 8 | 32 | 8 | 32 |
| Soporte Técnico y Documentación | 4 | 9 | 36 | 7 | 28 | 8 | 32 |
| Ponderado | | 191 | | 148 | | 186 | |

justificación:

|  |  |  |  |
| --- | --- | --- | --- |
| **Criterio** | **ClickUp** | **Trello** | **Miro** |
| Usabilidad + | Ofrece una gran cantidad de funcionalidades integradas como gestión de tareas, seguimiento de tiempo, documentos, formularios y dashboards. Su enfoque todo-en-uno lo hace ideal para equipos que buscan centralizar operaciones. | Trello es una herramienta visual centrada en tableros kanban. Tiene funcionalidades básicas, pero se pueden extender mediante "power-ups". No ofrece tantas funciones nativas como ClickUp. | Miro es una herramienta colaborativa visual que permite diagramas, mapas mentales y lluvia de ideas. Aunque no es una herramienta de gestión de tareas como tal, es muy poderosa para la planificación y diseño visual. |
| Personalización+ | Su interfaz puede resultar compleja al principio por la cantidad de funcionalidades, pero es intuitiva una vez comprendida. Ofrece una curva de aprendizaje manejable con ayuda integrada. | Muy fácil de usar desde el principio. Su diseño simple e intuitivo permite comenzar rápidamente, ideal para equipos sin experiencia técnica. | Interfaz visual muy amigable y fluida. Ideal para usuarios que prefieren trabajar de forma gráfica y colaborativa. |
| Funcionalidades+ | Alta capacidad de personalización: permite definir flujos de trabajo, tipos de tareas, vistas personalizadas y más. | Personalización limitada en la versión gratuita. Requiere "power-ups" para ampliar capacidades, lo que puede afectar la flexibilidad. | Ofrece flexibilidad en la creación de espacios de trabajo visuales, pero menos orientada a tareas o flujos estructurados. |
| Costo- | Su versión gratuita es generosa, pero muchas funciones avanzadas requieren planes pagos. La relación costo-beneficio sigue siendo favorable. | Trello ofrece un plan gratuito funcional, pero características clave requieren plan premium. Costos razonables en general. | Miro también ofrece plan gratuito, aunque limitado en número de tableros. Los planes pagos son necesarios para equipos grandes o funcionalidades más avanzadas. |
| Integración y Compatibilidad + | Alta compatibilidad con herramientas como Slack, Google Drive, Zoom, GitHub, entre muchas otras. Se integra fácilmente en ecosistemas complejos. | Tiene integraciones básicas a través de power-ups, aunque algunas están limitadas según el plan. | Muy buena integración con herramientas de colaboración como Microsoft Teams, Google Workspace, Slack, etc. Ideal para trabajo remoto. |
| Soporte Técnico y Documentación | Ofrece buen soporte técnico y una base de conocimiento extensa. Cuenta con plantillas, videos y comunidad activa. | Soporte y documentación adecuados, aunque no tan completos como los de ClickUp. La comunidad también aporta soluciones. | Excelente documentación y soporte para usuarios. Amplios recursos visuales, tutoriales y foros. |

**Conclusión**

Tras analizar y comparar las herramientas ClickUp, Trello y Miro en función de criterios clave como funcionalidades, usabilidad, personalización, costos, integración y soporte, se concluye que ClickUp representa la opción más completa y robusta para equipos que buscan centralizar su gestión de proyectos en una sola plataforma. Su alto nivel de funcionalidad, personalización y compatibilidad con otras herramientas la hacen ideal para entornos complejos y dinámicos.

**2. Herramientas para organización y manejo de los entregables**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Criterio** | **Peso** | **Trello (0-10)** | **Po** | **Jira (0-10)** | **Po** | **Google Drive (0-10)** | **Po** |
| **Usabilidad +** | 8 | 9 | 72 | 7 | 56 | 8 | 64 |
| **Personalización +** | 6 | 8 | 48 | 9 | 54 | 5 | 30 |
| **Funcionalidades +** | 8 | 7 | 56 | 9 | 72 | 6 | 48 |
| **Costo -** | 5 | 10 | -50 | 6 | -30 | 9 | -45 |
| **Integración y Compatibilidad +** | 6 | 8 | 48 | 9 | 54 | 9 | 54 |
| **Soporte Técnico y Documentación** | 4 | 7 | 28 | 8 | 32 | 8 | 32 |
| **Total, Ponderado en peso:** | |  | **202** |  | **238** |  | **183** |

**Interpretación:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Criterio** | **Jira** | **Trello** | **Google Drive** |
| Usabilidad | Interfaz profesional, requiere curva de aprendizaje. | Muy intuitiva y visual, fácil para nuevos usuarios. | Alta usabilidad para almacenamiento y edición colaborativa. |
| Personalización | Alta personalización en flujos de trabajo, tableros, permisos, etc. | Moderada, se pueden personalizar tableros y etiquetas. | Limitada, centrada en configuración de documentos y carpetas. |
| Funcionalidades | Completa suite para gestión ágil, seguimiento de incidencias, informes, automatizaciones. | Buena para gestión visual de tareas, limitada en seguimiento detallado o informes. | Fuerte para colaboración documental, pero no es una herramienta de gestión de proyectos. |
| Costo | Tiene versión gratuita con límites, versiones premium con funcionalidades completas, puede ser costosa para equipos grandes. | Versión gratuita funcional, versiones Business Class y Enterprise con mejores opciones. | Gratuito con cuenta Google; almacenamiento extra con costo. |
| Integración y Compatibilidad | Altamente compatible con herramientas de desarrollo (GitHub, Bitbucket, Confluence, etc.). | Se integra con múltiples apps (Slack, Google Drive, etc.), aunque con algunas limitaciones. | Integración nativa con productos de Google; también con Trello y otras apps mediante complementos. |
| Soporte Técnico y Documentación | Documentación muy completa y soporte profesional con distintos niveles según el plan. | Documentación clara y comunidad activa; soporte limitado en versión gratuita. | Soporte de Google y centro de ayuda detallado; excelente para funciones relacionadas a Drive. |

**Conclusión:**

Se concluye que Jira es la herramienta más completa y robusta, destacándose como la mejor opción para equipos que trabajan en entornos colaborativos debido a su alto nivel de funcionalidades, personalización y compatibilidad (238 puntos). Trello representa una alternativa balanceada, fácil de usar y especialmente efectiva para la gestión visual de tareas en equipos pequeños o proyectos personales, con un puntaje sólido de 202. Por su parte, Google Drive, aunque no está diseñada específicamente para la gestión de proyectos, sobresale en integración y colaboración en tiempo real, siendo útil como complemento para el seguimiento de documentos (183 puntos).

**3. Herramientas para Documentación de Requisitos**

**Matriz de Evaluación**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Criterio** | **Peso** | **Jira + Confluence (0-10)** | **Po** | **Azure DevOps (0-10)** | **Po** | **GitHub Issues + Wiki (0-10)** | **Po** | **Google Docs + Sheets (0-10)** | **Po** | **Microsoft Office 365**  **(0-10)** | **Po** |
| **Facilidad de Uso** | 4 | 8 | 32 | 8 | 32 | 10 | 40 | 10 | 40 | 8 | 32 |
| **Colaboración** | 4 | 10 | 40 | 10 | 40 | 8 | 32 | 10 | 40 | 8 | 32 |
| **Integración** | 5 | 10 | 50 | 10 | 50 | 6 | 30 | 6 | 30 | 10 | 50 |
| **Trazabilidad** | 5 | 10 | 50 | 10 | 50 | 6 | 30 | 4 | 20 | 3 | 15 |
| **Control de Versiones** | 4 | 8 | 32 | 10 | 20 | 10 | 40 | 6 | 24 | 6 | 24 |
| **Costo** | 3 | 6 | -18 | 4 | -12 | 10 | -30 | 10 | -30 | 6 | -18 |
| **Escalabilidad** | 4 | 10 | 40 | 10 | 40 | 8 | 32 | 6 | 24 | 8 | 32 |
| **Ponderado** | 29 (Suma de Po) |  | 244 |  | **254** |  | 214 |  | 178 |  | 185 |

**Justificación**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterio** | **Jira + Confluence** | **Azure DevOps** | **GitHub Issues + Wiki** | **Google Docs + Sheets** | **Microsoft Office 365** |
| **Facilidad de Uso** | Requiere familiarización con su estructura de proyectos y configuración. No es muy intuitiva para principiantes. | Interfaz moderna y relativamente sencilla para equipos técnicos. | Intuitiva para usuarios de GitHub. | Muy fácil de usar, familiar para la mayoría. | Familiar para muchos usuarios. |
| **Colaboración** | Alta colaboración entre equipos con historial de cambios y comentarios. | Muy buena colaboración con boards ágiles, work items y comentarios. | Buena colaboración mediante issues y comentarios. | Excelente en tiempo real. | Buena, pero no tan fluida como Google Docs. |
| **Integración** | Se integra bien con otros productos de Atlassian y herramientas de desarrollo. | Excelente integración con Git, CI/CD, VS Code, GitHub, etc. | Limitada comparada con otras herramientas más específicas. | Limitada con herramientas de desarrollo. | Limitada en entornos de desarrollo puro. |
| **Trazabilidad** | Excelente trazabilidad entre requisitos, tareas y bugs. | Alta trazabilidad entre requerimientos, código y pruebas. | Limitada sin plugins o configuraciones adicionales. | Manual, difícil de mantener en proyectos grandes. | Muy baja. |
| **Control de Versiones** | Limitado, depende de integraciones externas como Bitbucket. | Soporte nativo para Git. | Nativo en el repositorio. | Básico mediante historial de versiones. | Disponible con OneDrive, pero poco ágil. |
| **Costo** | Licencia paga, aunque con versión gratuita limitada. | Versión gratuita robusta para equipos pequeños. | Gratuito para la mayoría de los casos. | Gratuito. | Requiere suscripción. |
| **Escalabilidad** | Escalable para entornos empresariales complejos. | Altamente escalable para proyectos grandes y organizaciones. | Útil para proyectos medianos o personales. | Poco adecuado para proyectos complejos. | Más útil en entornos corporativos no técnicos. |

**Conclusión**

**Azure DevOps** es la herramienta más equilibrada, combinando trazabilidad, colaboración y control de versiones con una curva de aprendizaje accesible. Supera a herramientas más básicas como Google Docs o GitHub Wiki en funcionalidad técnica, y es más integrada y gratuita en comparación con Jira.

**4. Herramientas para Arquitectura y Diseño**

**Matriz de Evaluación**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Criterio** | **Peso** | **Lucidchart (0-10)** | **Po** | **Draw.io (diagrams.net) (0-10)** | **Po** | **MS Visio (0-10)** | **Po** | **PlantUML (0-10)** | **Po** | **Miro (0-10)** | **Po** |
| **Facilidad de Uso** | 4 | 10 | 40 | 8 | 32 | 6 | 24 | 4 | 16 | 10 | 40 |
| **Colaboración** | 3 | 10 | 30 | 8 | 24 | 6 | 18 | 6 | 18 | 10 | 30 |
| **Integración** | 5 | 8 | 40 | 8 | 40 | 8 | 40 | 10 | 50 | 6 | 30 |
| **Trazabilidad** | 5 | 6 | 30 | 4 | 20 | 2 | 10 | 8 | 40 | 4 | 20 |
| **Control de Versiones** | 4 | 8 | 32 | 10 | 40 | 6 | 24 | 10 | 40 | 6 | 24 |
| **Costo** | 3 | 6 | -18 | 10 | -30 | 4 | -12 | 10 | -6 | 6 | -18 |
| **Escalabilidad** | 4 | 8 | 32 | 8 | 24 | 8 | 32 | 10 | 40 | 8 | 32 |
| **Ponderado** | 28 (Suma de Po) |  | 204 |  | 180 |  | 170 |  | **228** |  | 176 |

**Justificación**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterio** | **Lucidchart** | **Draw.io (diagrams.net)** | **MS Visio** | **PlantUML** | **Miro** |
| **Facilidad de Uso** | Muy fácil e intuitivo. | Muy intuitivo, sin curva de aprendizaje. | Interfaz más técnica, puede ser compleja. | Requiere conocimientos técnicos (código). | Muy visual e intuitivo. |
| **Colaboración** | Colaboración en tiempo real con permisos detallados. | Permite edición en conjunto vía Google Drive o GitHub. | Limitada en comparación con herramientas web. | A través de repositorios Git. | Excelente para equipos creativos. |
| **Integración** | Buena con Google Drive, Slack, Confluence. | Compatible con plataformas comunes. | Funciona bien con Microsoft 365. | Excelente con herramientas de CI/CD y documentación. | Múltiples plugins disponibles. |
| **Trazabilidad** | Limitada. | Muy básica. | No especializada. | Alta, al estar basado en texto. | Limitada para uso técnico. |
| **Control de Versiones** | Versión básica de historial. | Depende del almacenamiento externo (Google Drive, GitHub). | Limitado, depende del almacenamiento. | Nativo al trabajar con Git. | Muy básico. |
| **Costo** | Tiene plan gratuito limitado. | Completamente gratuito. | Requiere licencia de pago. | Gratuito y open source. | Plan gratuito con restricciones. |
| **Escalabilidad** | Apto para equipos pequeños y medianos. | Apto para cualquier tipo de proyecto. | Apto para grandes empresas. | Muy alta. | Bueno para brainstorming, limitado para documentación técnica. |

**Recomendación: PlantUML**

**Conclusión**

PlantUML ofrece trazabilidad, control de versiones y automatización. Draw.io, por su parte, brinda facilidad y colaboración visual. La combinación cubre necesidades técnicas y de diseño conceptual, superando a herramientas con licencias costosas o con funciones limitadas como Visio o Miro.

**5. Herramientas para Informes de Pruebas**

**Matriz de Evaluación**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Criterio** | **Peso** | **Jira + Xray (0-10)** | **Po** | **GitHub Issues + Actions (0-10)** | **Po** | **Confluence + TestRail (0-10)** | **Po** | **Excel / Google Sheets (0-10)** | **Po** | **Allure Reports (0-10)** | **Po** |
| **Facilidad de Uso** | 4 | 8 | 32 | 8 | 32 | 6 | 24 | 10 | 40 | 6 | 24 |
| **Colaboración** | 3 | 10 | 30 | 8 | 24 | 8 | 24 | 6 | 18 | 6 | 18 |
| **Integración** | 5 | 10 | 50 | 10 | 50 | 8 | 40 | 4 | 20 | 10 | 50 |
| **Trazabilidad** | 5 | 10 | 50 | 8 | 40 | 10 | 50 | 2 | 10 | 8 | 40 |
| **Control de Versiones** | 4 | 8 | 32 | 10 | 40 | 6 | 24 | 4 | 16 | 8 | 32 |
| **Costo** | 3 | 4 | -12 | 2 | -6 | 4 | -12 | 2 | -6 | 2 | -6 |
| **Escalabilidad** | 4 | 10 | 40 | 8 | 32 | 10 | 40 | 4 | 16 | 8 | 32 |
| **Ponderado** | 28 (Suma de Po) |  | 234 |  | **242** |  | 202 |  | 144 |  | 220 |

**Recomendación: GitHub Issues + Actions**

**Justificación**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterio** | **Jira + Xray** | **GitHub Issues + Actions** | **Confluence + TestRail** | **Excel / Google Sheets** | **Allure Reports** |
| **Facilidad de Uso** | Requiere configuración avanzada. | Intuitiva para usuarios de GitHub. | Requiere familiarización. | Muy fácil. | Requiere configuración técnica. |
| **Colaboración** | Muy buena integración con equipos ágiles. | Muy buena, especialmente en proyectos open source. | Buena, especialmente en grandes equipos. | Google Sheets destaca en este aspecto. | Limitada. |
| **Integración** | Alto nivel con pipelines y otros plugins. | Altísima con CI/CD y testing automático. | Limitada fuera del ecosistema Atlassian. | Mínima. | Excelente con entornos de pruebas automatizadas. |
| **Trazabilidad** | Alta, enlaza pruebas con historias y errores. | Muy buena, desde código hasta ejecución. | Buena, pero con configuración adicional. | Muy baja. | Alta si se integra correctamente. |
| **Control de Versiones** | Limitado. | Totalmente integrado con Git. | Limitado. | Básico. | Depende del entorno de pruebas. |
| **Costo** | Requiere licencia para Xray. | Gratuito. | Licencia de TestRail es elevada. | Gratuito. | Gratuito. |
| **Escalabilidad** | Apto para grandes empresas. | Funciona en proyectos de todos los tamaños. | Alta. | Difícil de mantener en proyectos grandes. | Buena para entornos automatizados. |

**Conclusión**

GitHub Issues + Actions combina automatización, trazabilidad y colaboración sin costos adicionales. Es ideal para equipos modernos que buscan eficiencia en pruebas sin comprometer integración ni control.

**6. Herramientas para Documentación de Usuario**

**Matriz de Evaluación**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Criterio** | **Peso** | **GitBook (0-10)** | **Po** | **Confluence (0-10)** | **Po** | **MkDocs + GitHub Pages (0-10)** | **Po** | **Notion (0-10)** | **Po** | **Swagger/OpenAPI (0-10)** | **Po** |
| **Facilidad de Uso** | 4 | 10 | 40 | 8 | 32 | 6 | 24 | 10 | 40 | 6 | 24 |
| **Colaboración** | 3 | 8 | 24 | 10 | 30 | 6 | 18 | 10 | 30 | 6 | 18 |
| **Integración** | 4 | 8 | 32 | 8 | 32 | 10 | 40 | 6 | 24 | 10 | 40 |
| **Trazabilidad** | 3 | 6 | 18 | 6 | 18 | 6 | 18 | 4 | 12 | 8 | 24 |
| **Control de Versiones** | 3 | 8 | 24 | 6 | 18 | 10 | 30 | 4 | 12 | 8 | 24 |
| **Costo** | 3 | 6 | -18 | 6 | -18 | 2 | -6 | 8 | -24 | 2 | -6 |
| **Escalabilidad** | 4 | 8 | 32 | 10 | 40 | 8 | 32 | 8 | 32 | 8 | 32 |
| **Ponderado** | 24 (Suma de Po) |  | 170 |  | 170 |  | 186 |  | 150 |  | 194 |

**Recomendación: MkDocs + Swagger**

**Justificación**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterio** | **GitBook** | **Confluence** | **MkDocs + GitHub Pages** | **Notion** | **Swagger / OpenAPI** |
| **Facilidad de Uso** | Muy intuitivo, ideal para principiantes. | Requiere familiarización con su estructura. | Markdown sencillo, pero requiere conocimientos técnicos y manejo de Git. | Extremadamente fácil e intuitivo para cualquier usuario. | Técnica; requiere conocimientos sobre APIs y definición de endpoints. |
| **Colaboración** | Permite colaboración básica; limitada en el plan gratuito. | Alta colaboración con control de versiones y permisos. | Permite colaboración a través de repositorios Git, no en tiempo real. | Excelente colaboración en tiempo real con comentarios y menciones. | Puede compartirse, pero no permite colaboración simultánea en la edición. |
| **Integración** | Se conecta fácilmente con plataformas como Google Drive o Slack. | Se integra con herramientas del ecosistema Atlassian. | Alta integración con GitHub, CI/CD y despliegue automático (GitHub Pages). | Integración limitada con herramientas técnicas o pipelines de desarrollo. | Se integra con herramientas de desarrollo, APIs, pruebas automatizadas y Swagger UI. |
| **Trazabilidad** | Proporciona trazabilidad básica de cambios. | Buena trazabilidad si se configura correctamente. | Permite trazabilidad a través del control de versiones de Git. | Escasa; no ofrece herramientas específicas de seguimiento técnico. | Excelente trazabilidad técnica, especialmente para endpoints, respuestas y versiones. |
| **Control de Versiones** | Historial básico de edición, útil para cambios simples. | Ofrece historial, aunque limitado fuera de su ecosistema. | Control completo mediante Git, permite revertir o comparar versiones fácilmente. | Muy limitado, con opciones básicas de historial. | Control eficaz usando Git para mantener historial de documentación técnica. |
| **Costo** | Plan gratuito con funciones básicas, ampliables mediante suscripción. | Requiere licencia de pago para acceder a todas sus funcionalidades. | Gratuito y open source; se despliega sin costos adicionales en GitHub Pages. | Gratuito con funciones ampliadas en plan pago. | Completamente gratuito y de código abierto, sin restricciones de uso. |
| **Escalabilidad** | Escalable para proyectos pequeños y medianos. | Altamente escalable para empresas grandes con necesidades complejas. | Escalable y apto para documentaciones amplias y estructuradas técnicamente. | Escalable para documentación general, pero no para entornos técnicos avanzados. | Muy escalable y ampliamente adoptado en entornos empresariales y proyectos grandes. |

**Conclusión**

La combinación MkDocs + Swagger cubre la documentación tanto general como técnica. Permite control total de versiones, despliegue automático y documentación interactiva para usuarios técnicos y no técnicos, siendo una opción potente, gratuita y profesional.

**7.** **Herramientas para la revisión de metricas**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Criterio | Peso | Selenium  (0-10) | Po Selenium | Playwright  (0-10) | Po Playwright | Cypress  (0-10) | Po Cypress |
| Funcionalidad/características deseadas o requeridas | 5 | 4 | 20 | 4 | 20 | 3 | 15 |
| Facilidad de uso | 5 | 1 | 5 | 4 | 20 | 4 | 20 |
| Curva de aprendizaje | 4 | 1 | 4 | 4 | 16 | 4 | 16 |
| Compatibilidad/Integración a otras herramientas y aplicaciones | 5 | 4 | 20 | 4 | 20 | 3 | 15 |
| Costo/Licenciamiento | 3 | 4 | 12 | 4 | 12 | 2 | 6 |
| Escalabilidad | 5 | 4 | 20 | 4 | 20 | 3 | 15 |
| Flexibilidad/Capacidad de personalización | 3 | 4 | 12 | 4 | 12 | 4 | 12 |
| Lenguajes de programación soportados | 3 | 4 | 12 | 3 | 9 | 2 | 6 |
| Navegadores compatibles | 3 | 4 | 12 | 2 | 6 | 3 | 9 |
| Ponderado |  |  | 117 |  | 135 |  | 114 |

**Justificación:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Criterio** | **Selenium** | **Playwright** | **Cypress** |
| Funcionalidad/características deseadas o requeridas | Soporta pruebas automatizadas de múltiples tipos, incluyendo pruebas funcionales en diferentes navegadores y sistemas operativos. Puede manejar múltiples pestañas, ventanas, y trabajar con elementos dinámicos. Muy potente para pruebas complejas. | Puede hacer pruebas de interacción con APIs. Se pueden realizar pruebas paralelas y en múltiples navegadores al mismo tiempo. Permite subir archivos, manejar ventanas emergentes, cuadros de diálogo, etc. Permite pruebas en modo headless (sin interfaz) o con interfaz visible. | Tiene funciones modernas para pruebas de interfaz y testing de APIs, y permite ver paso a paso lo que ocurre en cada prueba. Sin embargo, su soporte para múltiples pestañas o ventanas es limitado. |
| Facilidad de uso | No es tan intuitivo como otras herramientas más modernas. Puede requerir configuración adicional y manejo de drivers de navegador. | Fácil de usar. El proceso de instalación es sencillo y posee una API moderna y limpia. | Destaca por su interfaz visual muy amigable, que permite ver las pruebas ejecutándose. Es intuitivo para usuarios con poca experiencia. |
| Curva de aprendizaje | La curva puede ser moderada a alta, especialmente si se requiere configuración avanzada o integración con otros frameworks. | La curva es baja. La herramienta es intuitiva y con ejemplos claros. | Es fácil comenzar a usar Cypress incluso sin mucha experiencia previa en testing, lo que reduce la curva de aprendizaje. |
| Compatibilidad/Integración a otras herramientas y aplicaciones | Alta compatibilidad con herramientas de CI/CD, frameworks de testing (JUnit, TestNG, etc.) y lenguajes de programación. Se integra fácilmente en ecosistemas complejos. | Funciona con frameworks como Pytest, Mocha, Jest y se integra nativamente con CI/CD (GitHub Actions, Azure DevOps, Jenkins) | Tiene integración con herramientas de CI/CD, pero algunas integraciones requieren planes pagos si se usan en su plataforma oficial (Dashboard). |
| Costo/Licenciamiento | Completamente gratuito y open source. | Completamente gratuito y open source. | Es open source y gratuito, pero algunas funciones avanzadas como el Dashboard y reportes automáticos tienen planes pagos. |
| Escalabilidad | Muy escalable, apto para proyectos pequeños, medianos y grandes en entornos empresariales. | Soporta ejecución en paralelo y containers por lo que tiene muy buena escalabilidad. | Funciona bien en proyectos medianos, pero su arquitectura basada en un solo navegador y algunas limitaciones técnicas. |
| Flexibilidad/Capacidad de personalización | Muy flexible, permite personalizaciones profundas en flujos de prueba y configuración. | Permite automatizar escenarios diversos y complejos con una API más simple. | Es flexible, aunque al estar más centrado en una estructura propia, puede ser menos adaptable a ciertas configuraciones externas. |
| Lenguajes de programación soportados | Java, Python, C#, Ruby, JavaScript, entre otros. | Python, JavaScript, TypeScript, C# y Java. | Está centrado en JavaScript y TypeScript. No soporta otros lenguajes. |
| Navegadores compatibles | Soporta una amplia gama de navegadores: Chrome, Firefox, Safari, Edge, Opera, entre otros, con soporte completo para pruebas cruzadas. | Firefox, Edge, Chrome y WebKit. | Funciona bien en Chrome, Firefox y Edge, pero aún no tiene soporte oficial completo para Safari y tiene limitaciones en pruebas cruzadas entre navegadores. |

**Conclusión**

Playwright es la mejor opción porque combina la potencia de Selenium con la facilidad de uso de Cypress. Ofrece una instalación sencilla, una API moderna y clara, soporte para pruebas paralelas, aunque no tiene tanto soporte de navegadores como Selenium, es compatible con los navegadores más populares. Comparativamente posee una curva de aprendizaje baja y una alta facilidad de uso y se integra nativamente con CI/CD. Esto lo hace más eficiente, escalable y versátil, cubriendo tanto proyectos simples como entornos empresariales sin las limitaciones técnicas de Cypress ni la complejidad de configuración de Selenium.

**Referencias**

[1] G2.com, "ClickUp Reviews, Pricing, and Features," *G2 Software Reviews*, 2023. [Online]. Available: <https://www.g2.com/products/clickup/reviews>

[2] A. Kumar and R. Singh, "Comparative Study of Project Management Tools: Jira, Trello, Asana and ClickUp," *International Journal of Computer Applications*, vol. 184, no. 5, pp. 12–17, May 2022.

[3] ClickUp, "ClickUp Docs & Product Features," *ClickUp Official Documentation*, 2024. [Online]. Available: <https://help.clickup.com>

[4] M. ProjectManagement.com, "ClickUp: A Comprehensive Project Management Tool," *Project Management Institute*, 2021. [Online]. Available: <https://www.projectmanagement.com>

[5] E. Gómez, "Análisis comparativo de herramientas de productividad en equipos remotos," *Revista de Ingeniería de Software y Sistemas*, vol. 14, no. 3, pp. 101–109, Sept. 2023.

[6] N. Sharma, “Jira vs Trello: Who wins the battle to project management?”, ProofHub, Feb. 26, 2025. [Online]. Available: <https://www.proofhub.com/articles/jira-vs-trello>

[7] Emerson College, “Google Drive to Increase Efficiency and Collaboration,” Emerson College Human Resources. [Online]. Available: <https://hr.emerson.edu/hc/en-us/articles/24945725961747-Google-Drive-to-Increase-Efficiency-and-Collaboration>

[8] Google, “Google Drive Overview,” Google Workspace, 2024. [Online]. Available: <https://workspace.google.com/products/drive/>

[9] «IEEE StD. 829 Test Plan Document |Professionalqa.com». <https://www.professionalqa.com/ieee-standard-829>

[10] «How to Write a Test Plan with the IEEE 829 Standard», *Reqtest*, 24 de agosto de 2023. <https://reqtest.com/en/knowledgebase/how-to-write-a-test-plan-2/>

[11] «IEEE 829 Test Plan example», *Google Docs*. <https://docs.google.com/document/d/1F1TUX5BkviRbw8auI2Xq2K2wyGTS-Ag14DRymiELmzc/edit?tab=t.0#heading=h.jd6vjs4lpb4w>

[12] Imaginary Cloud, «Software Architecture Documentation Best Practices and Tools», *Imaginary Cloud*. <https://www.imaginarycloud.com/blog/software-architecture-documentation>

[13] S. Murugesan, «A Comprehensive Guide to Software Architecture Documentation», *Document360*, 28 de julio de 2025. <https://document360.com/blog/software-architecture-documentation>

[6] The Agile Modeling (AM) Method, «UML Frames: Diagramming Style Guidelines», *The Agile Modeling (AM) Method - Effective Strategies For Modeling And Documentation*, 23 de noviembre de 2023. <https://agilemodeling.com/style/frame.htm>

[14] Ibrahimcanerdogan, «Chapter #7 UML Architecture Diagrams— Software Design and Architecture Specialization University of Alberta», *Medium*, 8 de noviembre de 2023. [En línea]. Disponible en: <https://ibrahimcanerdogan.medium.com/chapter-7-uml-architecture-diagrams-software-design-and-architecture-specialization-university-of-00eadcb9115d>

[15] «About the Unified Modeling Language Specification Version 2.5.1». <https://www.omg.org/spec/UML/2.5.1/About-UML>

[16] «UML and Design Documentation: Everything You Need to Know When Assessing UML and Design Documentation Skills». <https://www.alooba.com/skills/concepts/software-architecture-422/uml-and-design-documentation>

[17] K. Fakhroutdinov, «Unified Modeling Language (UML) description, UML diagram examples, tutorials and reference for all types of UML diagrams - use case diagrams, class, package, component, composite structure diagrams, deployments, activities, interactions, profiles, etc.» <https://www.uml-diagrams.org>

[18] “Xray Test Management for Jira | Atlassian Marketplace,” *Atlassian Marketplace*. <https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-jira>

[19] “Managing your work with GitHub Actions - GitHub Docs,” *GitHub Docs*. <https://docs.github.com/en/actions/tutorials/manage-your-work>

[20] “TestRail Test Management Integration for Jira | Atlassian Marketplace,” *Atlassian Marketplace*. <https://marketplace.atlassian.com/apps/1213701/testrail-test-management-integration-for-jira>

[21] “Allure Report — Open-source HTML test automation report tool.” <https://allurereport.org/>

[22] “GitBook – Build product documentation your users will love.” <https://www.gitbook.com/>

[23] Confluence, “Investment Management Solutions - Confluence Technologies,” *Confluence Technologies*, Jul. 22, 2025. <https://www.confluence.com/>

[24] M. Team, “Deploying your Docs - MKDocs.” <https://www.mkdocs.org/user-guide/deploying-your-docs/>

[25] “The AI workspace that works for you. | Notion,” *Notion*. <https://www.notion.com/>

[26] “OpenAPI Specification - Version 3.1.0 | Swagger.” <https://swagger.io/specification/>

[27] APIdog, “Selenium vs Playwright: ¿Cuál es mejor para tus pruebas automáticas?”, *APIdog Blog*, [En línea]. Disponible en: <https://apidog.com/es/blog/selenium-vs-playwright-3/>.

[28] *Playwright*, “Playwright Documentation”, *DevDocs.io*, [En línea]. Disponible en: <https://devdocs.io/playwright/>.

[29] Itequia, “Cypress: La herramienta que automatiza tus pruebas y garantiza la calidad de tus proyectos”, *Itequia Blog*, [En línea]. Disponible en: <https://itequia.com/es/cypress-la-herramienta-que-automatiza-tus-pruebas-y-garantiza-la-calidad-de-tus-proyectos/>.

# Sección 3 – SQA Tarea

## 3.1 – Tarea: Proceso de Evaluación de Análisis de Requisitos de Software

El propósito de esta tarea evaluativa del análisis de requisitos de software es generar, documentar y gestionar los requisitos de software; responder a solicitudes de aclaración, corrección o cambio; analizar los impactos derivados de los cambios en los requisitos establecidos; revisar las especificaciones de requisitos; y gestionar/revisar los procesos de análisis y cambio de requisitos de software. Este enfoque se alinea con la norma ISO/IEC/IEEE 29148:2018, que exige que los requisitos sean completos, consistentes, verificables y trazables durante todo el ciclo de vida del proyecto, y con las pautas de la IEEE 830‑1998, que definen un proceso formal para la aprobación y gestión de cambios.

**Herramienta de Gestión usada: Trello**  
Para soportar todo el proceso, se configuró el tablero “Análisis de Requisitos” con cinco listas principales:

* Recopilación de Requisitos
* Validación y Revisión
* Gestión de Cambios
* Trazabilidad
* Compromisos y Aprobaciones

**Tabla 4.1 Formulario – Roles y Responsabilidades (Matriz RACI)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Actividad** | **Responsable (R)** | **Consultado (C)** | **Informado (I)** |
| Definición inicial de requisitos | Analista de SQA | – | Analistas SQA |
| Validación y revisión de requisitos | Revisor de Calidad (SQA Reviewer) | Especialista Técnico (Arquitectura / Diseño) | Líder de Evaluación de Requisitos y Gestión |
| Gestión de cambios | Líder de Evaluación de Requisitos y Gestión | Analistas SQA | Revisor(es) de Calidad |
| Actualización de la Matriz de Trazabilidad | Analistas SQA | – | Revisor de Calidad |
| Aprobación final de requisitos | Líder de Evaluación de Requisitos y Gestión | – | Todos los miembros del equipo |

Se plasmarán los resultados de estas tareas de SQA utilizando el Formulario de Lista de Verificación de Auditoría de Procesos que se encuentra a continuación:

**Tabla 4.2 Formulario – Proceso de Análisis de Requisitos**

|  |
| --- |
| **Lista de verificación de análisis de requisitos del sistema** |
| **Análisis de Requisitos – Proyecto Actual  Fecha: 27 de julio de 2025  Elaborado por: Edgar Lorenzo y Anabely Abrego** |
| **Procedimientos:**  \_✅\_ 1. Los participantes correctos están involucrados en el proceso de análisis de requisitos del sistema para identificar todas las necesidades de los usuarios.  \_✅\_ 2. Los requisitos son revisados para determinar si son factibles de implementar, están claramente expresados y son consistentes.  \_✅\_ 3. Cambios identificados, revisados y cerrados.  \_✅\_ 4. El personal del proyecto involucrado en el proceso de análisis de requisitos del sistema está capacitado en los procedimientos y estándares necesarios aplicables a su área de responsabilidad para realizar correctamente el trabajo.  \_✅\_ 5. Los compromisos derivados de los requisitos asignados son negociados y acordados por los grupos afectados.  \_✅\_ 6. Los compromisos están documentados, revisados, aceptados, aprobados y comunicados.  \_✅\_ 7. Los requisitos asignados identificados como problemáticos son revisados con el grupo responsable del análisis de requisitos del sistema y documentos, y se realizan los cambios necesarios.  \_✅\_ 8. Procesos para definir, documentar y asignar requisitos documentados  \_✅\_ 9. Los requisitos están documentados, gestionados, controlados y rastreados (preferiblemente mediante una matriz). |
| **Evidencia:**  En la lista Recopilación de Requisitos de Trello se usan Custom Fields para asignar y registrar rol (Propietario, Analista, Usuario final) en cada tarjeta.  En la lista Validación y Revisión existen checklists de “Claridad”, “Factibilidad” y “Consistencia”, además de etiquetas (Pendiente / En Revisión / Aprobado / Rechazado).  El historial de Trello permite rastrear cambios por usuario y fecha.  Se adjuntan actas de reuniones como soporte a compromisos aprobados.  Checklists por usuario confirman capacitación en análisis y estándares aplicables. |

## – Tarea: Proceso de Evaluación del Diseño

Esta tarea tiene como finalidad asegurar que todas las actividades relacionadas con el diseño del sistema se ejecuten conforme a lo establecido en el plan de pruebas, y que los resultados obtenidos permitan concluir si el sistema cumple con los requisitos funcionales y no funcionales definidos. El enfoque está alineado con las mejores prácticas de calidad según el estándar IEEE 730-1998, adaptadas al entorno académico del presente proyecto.

La revisión del diseño buscará confirmar que el código alineado con la arquitectura aprobada, mantenga trazabilidad completa (requisitos-código-pruebas) y que las pruebas unitarias estén debidamente ejecutadas y documentadas.

**Tipos de Prueba Considerados**

- Pruebas unitarias: validan módulos individuales del sistema.  
- Pruebas de integración: verifican la correcta interacción entre componentes.  
- Pruebas del sistema: comprueban el comportamiento global frente a los requisitos.  
- Pruebas de aceptación: simulan escenarios de uso reales para validar la experiencia del usuario.  
- Pruebas de regresión: aseguran que cambios recientes no afectaron funcionalidades existentes.

**Tabla 5 Formulario – Proceso de Diseño**

|  |
| --- |
| **Lista de verificación del proceso de diseño** |
| **Proceso de Verificación- Proyecto Actual**  **Fecha:  28 de julio de 2025**  **Elaborado por: Melanie Martínez y Lisbeth Magallón** |
| **Procedimientos:**  \_\_✅ \_\_ Los documentos de diseño del sistema y la matriz de trazabilidad están preparados, actualizados y consistentes.  \_\_✅ \_\_ Los documentos relevantes de diseño del sistema se actualizan basándose en cambios aprobados de los requisitos.  \_\_✅ \_\_ Las revisiones de diseño (revisiones por pares) evalúan el cumplimiento del diseño con los requisitos, identifican defectos en el diseño y evalúan alternativas.  \_\_✅ \_\_ El personal del proyecto involucrado en el proceso de análisis de requisitos del sistema está capacitado en los procedimientos y estándares necesarios aplicables a su área de responsabilidad para realizar correctamente el trabajo.  \_\_✅ \_\_ Los compromisos derivados de los requisitos asignados son negociados y acordados por los grupos afectados.  \_\_✅ \_\_ Los compromisos están documentados, revisados, aceptados, aprobados y comunicados.  \_\_✅ \_\_ Los requisitos asignados identificados como problemáticos son revisados con el grupo responsable del análisis de requisitos del sistema y documentos, y se realizan los cambios necesarios.  \_\_✅ \_\_ Se siguen y documentan los procesos prescritos para definir, documentar y asignar requisitos.  \_\_✅ \_\_ Los requisitos están documentados, gestionados, controlados y rastreados (preferiblemente mediante una matriz).  \_\_✅ \_\_ Los siguientes documentos se someten a revisión por pares durante esta fase del desarrollo:  \_\_✅ \_\_ Documento de Diseño de Software  \_\_✅ \_\_ Documento de Diseño de Interfaces  \_\_✅ \_\_ Manual de Programadores de Software |
| **Evidencias:**  Documento de Diseño de Software (DDS) y Matriz de Trazabilidad alojados en Google Drive con enlace referenciado en tarjetas Trello.  Versionado del DDS con control de cambios por fecha y autor.  Revisiones por pares con comentarios de mejora, checklist completadas en Trello.  Interfaces y manuales técnicos en documentos compartidos y revisados colaborativamente.  Estas herramientas facilitaron la colaboración entre los miembros del equipo en tiempo real. |

## – Tarea: Evaluación del Proceso de Implementación

La evaluación del proceso de implementación para el proyecto actual se centrará en verificar que el código desarrollado cumpla con el diseño aprobado, que exista trazabilidad entre los requisitos, el código y las pruebas, y que se hayan ejecutado y documentado correctamente las pruebas unitarias. Se revisará también que los hallazgos encontrados durante la implementación hayan sido registrados y solucionados.

Para esta tarea, se utilizarán documentos clave del proyecto que deben ser validados mediante listas de verificación específicas. Estos documentos permiten asegurar que cada componente implementado cumple con lo que se definió en etapas anteriores.

**Tabla 6 Formulario de Herramientas utilizadas para la evaluación de implementación**

|  |  |  |
| --- | --- | --- |
| Herramientas utilizadas para la evaluación de implementación | | |
| Tipo de Herramienta | Herramientas Seleccionadas | Uso dentro del Proyecto |
| Trazabilidad de Requisitos | Trello, Jira | Seguimiento de cumplimiento funcional y visualización de dependencias. |
| Gestión de Cambios | Jira, PowerAutomate | Seguimiento de tareas, acciones correctivas y automatización de flujos. |
| Control de Código Fuente | GitHub | Revisión de commits, historial de cambios y control de versiones. |
| Pruebas Unitarias | Junit | Ejecución de pruebas automatizadas por módulo. |
| Bitácora de Revisiones | Google Drive / Git Logs | Registro de revisiones entre pares y observaciones. |

**Documento de Diseño de Software**

Uso: Referencia estructural. No se valida el diseño en sí, sino su fidelidad en la implementación.

Checklist:

**Tabla 6.1 Checklist de verificación de implementación del diseño**

|  |
| --- |
| * ¿El código implementado respeta los módulos, componentes o clases definidos en el diseño? |
| * ¿Las interfaces entre componentes están implementadas según lo indicado? |
| * ¿Se mantienen las dependencias y jerarquías establecidas? |
| * ¿Los nombres, estructuras y responsabilidades de las clases coinciden? |
| * ¿Se respetaron los patrones o arquitecturas definidos (por ejemplo, MVC, capas, etc.)? |

**Matriz de Trazabilidad de Requisitos**

Uso: Confirmar correspondencia entre requisitos y código implementado.

Checklist:

**Tabla 6.2 Checklist de verificación de requisitos y código**

|  |
| --- |
| * ¿Cada requisito funcional tiene una unidad de código (función, clase, módulo) que lo implemente? |
| * ¿La trazabilidad directa entre código y requerimiento está clara? |
| * ¿Hay evidencia (en commits, documentación o pruebas) de que cada requerimiento fue implementado? |
| * ¿No se han implementado funcionalidades no trazadas a requisitos? |

**Especificación de Requisitos del Software**

Uso: Validación funcional. Se usa como base para verificar si lo que se implementó cumple con lo que el sistema debía hacer.

Checklist:

**Tabla 6.3 Checklist de validación funcional según SRS**

|  |
| --- |
| * ¿Cada funcionalidad implementada corresponde a lo descrito en la SRS (Especificación de Requisitos de Software) ? |
| * ¿Se respetaron las restricciones o reglas de negocio? |
| * ¿Se implementaron correctamente los casos de uso esperados? |
| * ¿Hay omisiones de funcionalidades críticas descritas en la SRS? |
| * ¿Las entradas y salidas del sistema son coherentes con lo especificado? |

**Tabla 6.4 Formulario – Proceso de Implementación de Software**

|  |
| --- |
| **Lista de verificación del proceso de implementación de software** |
| **Proceso de implementación – Proyecto Actual  Fecha: 28 de julio de 2025  Elaborado por: Rafaela Candanedo** |
| **Procedimientos:**  ✅ 1. El código y la matriz de trazabilidad están preparados, actualizados y son consistentes con los cambios aprobados en los requisitos de software. ✅ 2. Las revisiones de código (peer review) evalúan el cumplimiento del código respecto al diseño aprobado, identifican defectos y proponen alternativas. ✅ 3. Los cambios en el código están identificados, revisados y se les da seguimiento hasta su cierre. ✅ 4. Las pruebas unitarias del software se realizan conforme a los estándares y procedimientos aprobados. ✅ 5. Se documentan los criterios de aprobación de pruebas unitarias y se registra el cumplimiento correspondiente. |
| **Evidencia:**  Historial de pull requests y GitHub Actions (o tickets de Jira) muestran seguimiento y cierre de cada cambio.  Registros de peer reviews y comentarios de reviewers disponibles en la Bitácora de Revisiones de GitHub. |

## 3.4 – Tarea: Evaluación del Proceso de Pruebas

Esta tarea tiene como finalidad asegurar que todas las actividades relacionadas con la verificación y validación del sistema se ejecuten conforme a lo establecido en el plan de pruebas, y que los resultados obtenidos permitan concluir si el sistema cumple con los requisitos funcionales y no funcionales definidos. El enfoque está alineado con las mejores prácticas de calidad según el estándar IEEE 730-1998, adaptadas al entorno del presente proyecto.

**Objetivo**

Garantizar que el software desarrollado ha sido sometido a pruebas rigurosas que verifiquen su correcto funcionamiento, estabilidad, cumplimiento de requisitos y preparación para ser entregado al usuario final. Esto incluye la validación de las interfaces, el rendimiento bajo condiciones normales y excepcionales, y la experiencia de usuario.

**Tipos de Prueba Considerados**

- Pruebas unitarias: validan módulos individuales del sistema.  
- Pruebas de integración: verifican la correcta interacción entre componentes.  
- Pruebas del sistema: comprueban el comportamiento global frente a los requisitos.  
- Pruebas de aceptación: simulan escenarios de uso reales para validar la experiencia del usuario.

**Criterios de Éxito**

El sistema se considerará aprobado en la fase de pruebas si al menos el 90% de los casos de prueba definidos se ejecutan con éxito, y los errores encontrados han sido clasificados como menores o corregidos adecuadamente.

**Tabla 7. Formulario de Verificación de Procesos – Proceso de Pruebas**

|  |
| --- |
| **Lista de verificación del proceso de pruebas** |
| **Proceso de pruebas – Proyecto Actual**  **Fecha: 28 de julio de 2025**  **Elaborado por: Eriol Tuñon, Kazim Jesse** |
| **Procedimientos:**  \_✅\_\_ Los casos de prueba están definidos.  \_\_x\_\_ Se siguen los casos de prueba definidos.  \_✅\_\_\_ Están definidos los criterios de aprobación o rechazo de las pruebas.  \_\_x\_\_ Existe un plan de pruebas aprobado y una descripción de las pruebas.  \_✅\_\_\_ El entorno de pruebas, incluyendo tanto los requerimientos de hardware como de software, está configurado según lo requerido por el plan de pruebas.  \_\_✅\_\_ Cada caso de prueba en la descripción de pruebas es ejecutado.  \_\_✅\_\_ Los resultados de las pruebas se registran en un informe de pruebas.  \_\_✅\_\_ Todos los casos de prueba son ejecutados sobre el sistema. |

**Herramientas Utilizadas**

Para la planificación, ejecución y documentación de las pruebas, se utilizaron herramientas como Google Docs para registrar los planes y resultados, Trello para asignar y visualizar tareas de prueba, y Google Drive como repositorio central de evidencia (capturas, archivos, informes). Estas herramientas facilitaron la colaboración entre los miembros del equipo en tiempo real.

**Riesgos Considerados**

- Cobertura incompleta de pruebas por falta de tiempo o definición de escenarios.  
- Ambigüedad en requisitos que genera casos de prueba poco claros.  
- Registro inadecuado de resultados o fallos que dificulta la trazabilidad.  
- Correcciones tardías que no permiten validar completamente los cambios.

## 3.5 – Tarea: Evaluación del Proceso de Gestión de Despliegue y Liberación

El propósito de esta tarea evaluativa del proceso de gestión de liberación y despliegue es asegurar que la entrega del software a producción se realice de manera controlada, cumpliendo con todos los criterios técnicos y de calidad, y minimizando los riesgos operacionales. La evaluación busca validar que las actividades previas al despliegue, el proceso mismo y el seguimiento posterior se desarrollen según lo establecido.

Este enfoque se alinea con las prácticas de calidad establecidas en el estándar IEEE 828-2012, que promueve una gestión formal y trazable del proceso de liberación del software, incluyendo auditorías, validaciones del entorno, procedimientos de rollback y monitoreo post-deployment.

**Herramientas De Gestión usadas:**

Se utilizaron estas herramientas para apoyar las actividades del proceso de liberación: GitHub para la generación de versiones y empaquetado del software, Trello y Jira para el seguimiento de tareas y cambios aprobados, Google Docs para la documentación de procedimientos y validaciones, y pipelines de integración continua para automatizar el proceso de build y deployment.

**Tabla 8. Formulario de Verificación de Procesos – Proceso de Gestión de Liberación/Despliegue**

|  |
| --- |
| **Gestión de liberación / Lista de verificación de despliegue** |
| **Proceso de gestión de liberación – Proyecto Actual**  **Fecha: 28 de julio de 2025**  **Elaborado por: Felix Beitia y Cristopher Quintero** |
| **Procedimientos:**  \_✅\_\_\_ El software se genera desde la biblioteca de software de acuerdo con el plan de desarrollo.  \_✅\_\_\_ La estrategia para el entorno de producción está bien definida y diseñada.  \_✅\_\_\_ El entorno de producción está configurado correctamente para el despliegue antes del mismo.  \_✅\_\_\_ Las solicitudes de cambio son asumidas por los equipos de desarrollo una vez aprobadas por la gestión del programa.  \_✅\_\_\_ El proceso de solicitudes de cambio se sigue correctamente. |
| **Evidencia:**   * Confirmación del *build* automatizado sin errores, generado desde la rama principal en GitHub. * Documento de estrategia de despliegue y configuración del entorno almacenado en Google Docs. * Validación técnica del entorno antes del deployment documentada por el equipo QA. * Registro de aprobaciones de cambios y sign-off del equipo en Trello y Jira. * Verificación de integridad del paquete mediante checksums y firmas digitales. * Planes de rollback y pruebas de smoke validadas en entorno de producción. |

**Sección 4 – Documentación**

Esta sección establece los requisitos mínimos para la documentación de todos los artefactos de software y resultados relacionados. El propósito es garantizar una trazabilidad clara, facilitar la validación de entregables, asegurar el cumplimiento de los requisitos, y servir como insumo para mantenimiento futuro del software.

Para asegurar que el proceso de implementación de software satisfaga los estándares de calidad establecidos, se definen las siguientes prácticas de documentación como requerimientos mínimos:

**4.1 – Documento de Requisitos de Software**

Este documento establece todos los requisitos funcionales, atributos de calidad y restricciones del sistema. Constituye la base contractual entre los stakeholders y el equipo de desarrollo.

Lista de Verificación de SQA para la Documentación – Documento de Requisitos de Software

|  |
| --- |
| **LISTA DE VERIFICACIÓN DEL DOCUMENTO DE REQUISITOS DE SOFTWARE** |
| **Proyecto: SQA**  **Fecha: 28-07-2025**  **Elaborado por: Danna Dawkins, Abigail Koo, María Quiñones** |
| **Procedimientos:**  \_\_X\_ El documento describe todos los requisitos funcionales, atributos de calidad y restricciones con un identificador único.  \_\_X\_ Los requisitos funcionales han sido revisados y aprobados por el Gerente de Aplicaciones (o equivalente) y por el área de negocio.  \_\_X\_ El documento garantiza que todas las notas relevantes tomadas durante el proceso de entrevistas hayan sido formalmente registradas o referenciadas en otro documento. |

**4.2 – Informes de Pruebas de Software**

La documentación de pruebas es fundamental para garantizar que el sistema cumple con los requisitos funcionales y de calidad definidos. Su objetivo es registrar de forma estructurada los resultados de las pruebas realizadas, facilitando así la trazabilidad de errores, la evaluación de la cobertura y la mejora continua del producto.

Esta sección describe cómo estructurar correctamente los informes de prueba, de acuerdo con buenas prácticas y siguiendo como referencia el estándar **IEEE 829 - Standard for Software and System Test Documentation**.

**Contenido recomendado**

A continuación, se presentan los elementos esenciales que debe incluir un informe de pruebas:

**1. Plan de pruebas (resumen del enfoque)**

Aunque no es el plan completo, se recomienda que el informe incluya una breve introducción que resuma la estrategia de pruebas aplicada.

**Debe incluir:**

* Objetivos de las pruebas (ej.: validar funcionalidades principales, identificar fallos de integración).
* Tipo de pruebas realizadas: unitarias, de integración, funcionales, de rendimiento, etc.
* Ambiente de pruebas (ej.: navegador, base de datos, sistema operativo).
* Herramientas utilizadas (si aplica).

**Ejemplo:**

*"Se realizaron pruebas funcionales y de integración en el entorno de pruebas controlado, utilizando el navegador Chrome versión 122 y una base de datos PostgreSQL en servidor local."*

**2. Casos de prueba (Test Cases)**

Los casos de prueba documentan cómo se prueba cada funcionalidad. Sirven como evidencia de que el sistema fue evaluado de forma sistemática y repetible.

**Debe incluir:**

* **ID del caso de prueba:** Ej. TC-01, TC-02, etc.
* **Descripción:** Qué se está probando.
* **Precondiciones:** Estado inicial necesario para ejecutar la prueba.
* **Pasos a seguir:** Instrucciones para ejecutar la prueba.
* **Resultado esperado:** Qué se espera que ocurra.
* **Resultado real:** Qué ocurrió realmente al ejecutar la prueba.
* **Estado:** Aprobado, Fallido, Bloqueado.

**Ejemplo:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Descripción** | **Resultado Esperado** | **Resultado Real** | **Estado** |
| TC-01 | Iniciar sesión válida | Acceso al panel principal | Acceso correcto | Aprobado |
| TC-02 | Contraseña incorrecta | Mostrar mensaje de error | Muestra mensaje | Aprobado |
| TC-03 | Cerrar sesión | Redirige a login | No redirige | Fallido |

**3. Registro de errores o incidencias**

Cada falla o comportamiento no esperado detectado durante las pruebas debe registrarse como una incidencia, con suficiente detalle para que pueda ser reproducida y corregida.

**Debe incluir:**

* Código o ID del error (Bug-01, INC-04, etc.)
* Descripción del error.
* Nivel de severidad (Alta, Media, Baja).
* Fecha de detección.
* Estado actual (pendiente, en revisión, resuelto).
* Responsable asignado (opcional).

**Ejemplo:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Descripción** | **Severidad** | **Estado** |
| INC-01 | Botón “Guardar” no responde en Firefox | Alta | Resuelto |
| INC-02 | Ortografía incorrecta en mensaje de error | Baja | Pendiente |

**4. Métricas de calidad (opcional)**

Para proyectos más avanzados, es buena práctica incluir métricas que midan la efectividad del proceso de pruebas.

**Ejemplos de métricas:**

* Porcentaje de casos aprobados.
* Número total de errores detectados.
* Severidad promedio de errores.
* Tasa de retrabajo por módulo.

Estas métricas pueden presentarse en tablas o gráficos, si el equipo desea hacer un análisis más visual.

**5. Observaciones y recomendaciones finales**

Cierre del documento donde se mencionan hallazgos importantes, funcionalidades críticas que requieren atención y sugerencias para mejorar la cobertura de pruebas o evitar errores en versiones futuras.

**4.3 – Arquitectura y Diseño de Software**

Esta sección tiene como objetivo establecer cómo los desarrolladores deben documentar la estructura técnica del sistema, incluyendo la arquitectura general y los principales componentes de diseño. Esta documentación debe servir como guía para otros desarrolladores, testers y responsables técnicos, asegurando que todos entienden cómo está construido el sistema y por qué se tomaron ciertas decisiones técnicas.

La documentación debe seguir principios de modelado estandarizados, como los definidos en **UML (Unified Modeling Language)**, y estar alineada con los requisitos funcionales del sistema.

**Contenido recomendado**

A continuación, se indican los elementos clave que deben integrarse en la documentación de arquitectura y diseño:

**1. Descripción general de la arquitectura**

Presenta una vista de alto nivel sobre cómo está organizado el sistema. Esta vista debe enfocarse en componentes principales y cómo interactúan entre sí.

**Debe incluir:**

* Descripción de los módulos principales (ej.: frontend, backend, base de datos).
* Tecnología usada en cada módulo (opcional si el público objetivo es técnico).
* Diagrama de arquitectura general: componentes, flujos de datos, comunicación entre capas.

**Ejemplo:**

*"El sistema se organiza en una arquitectura de tres capas: presentación (interfaz de usuario), lógica de negocio (API REST) y persistencia de datos (base de datos relacional)."*

**2. Diagramas de diseño (UML u otros)**

Los diagramas ayudan a representar visualmente el diseño del sistema. Deben estar etiquetados y explicados de forma que un lector pueda entenderlos sin necesidad de consultar el código fuente.

**Diagramas recomendados:**

* **Diagrama de casos de uso:** Muestra las funcionalidades clave desde la perspectiva del usuario.
* **Diagrama de clases:** Representa las entidades del sistema y sus relaciones.
* **Diagrama de secuencia:** Describe el flujo de mensajes entre objetos o capas durante una interacción.
* **Diagrama de componentes o paquetes (opcional):** Útil para proyectos grandes.

Cada diagrama debe estar acompañado de:

* Un título descriptivo.
* Una breve explicación de su propósito.
* Leyendas o notas que aclaren elementos no evidentes.

**3. Decisiones de diseño**

Sección narrativa donde se explican las decisiones técnicas importantes que se tomaron durante el diseño del sistema.

**Debe incluir:**

* Por qué se eligieron ciertos patrones de diseño (ej.: MVC, Repositorio, Singleton).
* Elección de tecnologías o frameworks.
* Justificación de divisiones modulares (ej.: microservicios vs. monolito).
* Compromisos o limitaciones asumidas.

**Ejemplo:**

*"Se adoptó el patrón MVC para separar claramente la lógica de negocio de la interfaz. Esto facilita el mantenimiento y la escalabilidad del sistema en futuras versiones."*

**4. Trazabilidad con requisitos**

Idealmente, se debe establecer una relación entre los requisitos definidos y los elementos del diseño que los cumplen. Esto puede hacerse mediante una **matriz de trazabilidad** o referencias cruzadas.

**Ejemplo:**

* Requisito R-03 → Se implementa mediante la clase ControladorUsuarios y se visualiza en el diagrama de casos de uso UC-02.

**5. Recomendaciones finales (opcional)**

Puede incluir sugerencias para mantenimiento futuro, posibles mejoras a nivel arquitectónico o advertencias sobre limitaciones técnicas identificadas durante el diseño.

**4.4 – Documentación de Usuario**

**Contenido recomendado**

A continuación, se describen en detalle los apartados fundamentales que debe incluir la documentación dirigida a usuarios finales. Esta estructura puede adaptarse según el tipo de sistema desarrollado (aplicación web, móvil, de escritorio, etc.), pero debe mantenerse como base para asegurar la calidad y usabilidad del producto.

**1. Introducción y propósito**

Este apartado debe presentar brevemente al lector el objetivo de la aplicación o sistema, el público al que está dirigido, y la finalidad del documento. Sirve como una vista general que contextualiza lo que el lector encontrará a lo largo del manual.

**Debe incluir:**

* Una breve descripción del sistema (sin entrar en detalles técnicos).
* La utilidad principal del producto.
* A quién está dirigido el documento: ¿usuarios finales?, ¿administradores?, ¿usuarios técnicos?, ¿mixtos?
* Alcance del manual: qué aspectos del sistema cubre y qué no cubre.

**Ejemplo:**

*"Este documento está destinado a usuarios administrativos de la aplicación de inventario. Su propósito es guiar paso a paso el uso de las principales funcionalidades relacionadas con el registro, búsqueda y edición de productos en el sistema."*

**2. Requisitos previos o configuración inicial**

En este punto se detalla todo lo que el usuario necesita saber o hacer antes de comenzar a usar el sistema correctamente. Esto puede incluir tanto aspectos técnicos como conocimientos previos necesarios.

**Debe incluir:**

* Requisitos mínimos de hardware y software (ej.: navegador recomendado, sistema operativo compatible).
* Pasos de instalación o acceso (registro de cuenta, descarga del sistema, acceso a una URL).
* Configuraciones necesarias (ajustes iniciales, permisos, conexión a red, etc.).

**Ejemplo:**

* "Es necesario tener instalado el navegador Google Chrome (versión 90 o superior)."
* "Para acceder al sistema, el usuario debe contar con un correo electrónico institucional activo."
* "La primera vez que se inicia sesión, el sistema solicitará cambiar la contraseña predeterminada."

**3. Guía de uso paso a paso**

Esta sección constituye el núcleo del documento. Aquí se describe con claridad cómo utilizar el sistema, dividida por funcionalidades o módulos. Cada instrucción debe estar redactada en lenguaje sencillo y lógico, acompañada de ejemplos o escenarios de uso cuando sea necesario.

**Debe incluir:**

* Un título claro para cada función explicada (ej.: “Registrar un nuevo cliente”, “Exportar reportes”).
* Pasos numerados y secuenciales, iniciando desde la acción más básica.
* Indicaciones visuales: mencionar nombres de botones, íconos o pestañas específicas.
* Condiciones o advertencias importantes (por ejemplo, “el campo es obligatorio”, “no se puede editar después de guardar”).

**Ejemplo:**

**Funcionalidad: Crear un nuevo registro de usuario**

1. Ingrese al sistema con su usuario y contraseña.
2. Diríjase al menú lateral y haga clic en “Usuarios”.
3. Seleccione la opción “Nuevo Usuario”.
4. Complete los campos requeridos: nombre, correo electrónico, rol.
5. Haga clic en “Guardar” para finalizar el proceso.

**Nota:** El sistema enviará un correo automático con las credenciales al usuario registrado.

**4. Capturas de pantalla o ejemplos visuales**

Las imágenes ayudan a reforzar las instrucciones escritas. Cada procedimiento importante debe ir acompañado, cuando sea posible, de una o más capturas de pantalla ilustrativas. También se pueden usar diagramas, íconos destacados o gifs animados si la plataforma lo permite.

**Debe incluir:**

* Capturas nítidas, con buena resolución.
* Indicadores visuales (círculos, flechas o resaltados) para señalar botones, formularios, opciones de menú, etc.
* Breves títulos o descripciones debajo de cada imagen.

**Ejemplo:**

Imagen 1 – Menú de navegación principal.

*La opción “Configuración” se encuentra en el extremo superior derecho del panel.*

**5. Resolución de problemas (FAQ o troubleshooting)**

Este apartado anticipa dudas o errores comunes que podrían tener los usuarios. Brinda soluciones prácticas que el usuario pueda aplicar sin asistencia técnica, y ayuda a reducir consultas recurrentes al equipo de soporte.

**Debe incluir:**

* Preguntas frecuentes (FAQ) sobre el uso general del sistema.
* Mensajes de error comunes y sus posibles causas.
* Soluciones o recomendaciones concretas.

**Ejemplo:**

**Pregunta:** ¿Qué hago si olvidé mi contraseña?

**Respuesta:** Haga clic en “¿Olvidaste tu contraseña?” en la pantalla de inicio de sesión. El sistema enviará un enlace de recuperación a su correo electrónico registrado.

**6. Glosario (opcional)**

Cuando el sistema utiliza términos técnicos o propios del dominio de negocio (por ejemplo: “token”, “backend”, “checkout”, “cliente tipo B”), es recomendable incluir un glosario al final del documento que explique el significado de estos conceptos en lenguaje simple.

**Debe incluir:**

* Términos ordenados alfabéticamente.
* Definiciones breves y directas.

**Ejemplo:**

**Token de seguridad:** Cadena de caracteres única que se utiliza para validar el acceso del usuario por un periodo de tiempo limitado.

# Sección 5 – Medición de la Calidad del Software

**Tabla 9-1. Métricas Específicas por Etapa del SLDC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Fase del CVDS** | **Métrica** | **Límite de Tolerancia Inferior** | **Límite de Tolerancia Superior** | **Frecuencia de Medición** | **Acción Correctiva** |
| Definición | Porcentaje de defectos cuyo origen son errores en los requisitos | 2.5% | 5% | Después del lanzamiento del producto software. | Revisión de requerimientos con el cliente |
| Diseño | Porcentaje de defectos cuyo origen son errores de diseño. | 5% | 10% | Después del lanzamiento del producto software. | Revisión de arquitectura y revalidación del modelo |
| Desarrollo | Número de errores corregidos por cada error detectado en un commit. | 95% | 100% | Por sprint | Refactorización del código y revisión de pull requests |
| Pruebas | Porcentaje de casos de prueba exitosos | 88% | 100% | Al finalizar la fase de pruebas | Añadir pruebas unitarias adicionales / corrección de errores |
| Despliegue | Número de defectos identificados antes del despliegue | 5 | 9 | Antes del lanzamiento del producto software. | Revisión de checklist de QA / pruebas de regresión |

**Tabla 9-2. Métricas Globales de SQA**

|  |  |  |  |
| --- | --- | --- | --- |
| **Métrica Global de SQA** | **Límite de Tolerancia Inferior** | **Límite de Tolerancia Superior** | **Acción Correctiva** |
| Eficiencia en la Detección de Defectos | 93% | 100% | Evaluar cobertura de pruebas y ajustar test plan |
| Eficiencia en la Eliminación de Defectos | 95% | 100% | Revisión de procesos de corrección y retesting |
| Tiempo Promedio de Reparación | 2 horas | 8 horas | Establecer mecanismos de respuesta rápida (hotfix) |

# Sección 6 – Capacitación

La capacitación para el proyecto Sistema de Gestión de Calidad para Procesos Académicos (SGCPA) está orientada a garantizar que todos los miembros del equipo involucrados en actividades de aseguramiento de la calidad de software (SQA) posean las habilidades necesarias para desempeñar sus roles con eficacia. A partir del documento Roles Involucrados y las necesidades técnicas del proyecto, se identificaron las siguientes actividades de formación:

**Table 10. Matriz de Capacitación para SQA**

|  |  |  |  |
| --- | --- | --- | --- |
| **TAREA** | **HABILIDADES REQUERIDAS** | **TIPO** | **FUENTE** |
| Revisión de Código | Buenas prácticas de codificación, revisión por pares, trazabilidad | En el puesto de trabajo | Líder de Desarrollo / Coordinador QA |
| Revisión de Documentación | Redacción técnica, uso de plantilla IEEE, claridad en requisitos | Aula virtual / Taller | Coordinador de Calidad / Tutor |
| Auditorías de Proceso | Normas IEEE 730, validación de procesos, listas de verificación | En aula / simulación | Coordinador de Calidad / Docente |
| Pruebas | Caja negra y blanca, casos de prueba, criterios de validación | En el puesto de trabajo | Líder de QA / Documentos de prueba |
| Gestión de SQA | Planificación de calidad, seguimiento de entregables, gestión de tareas | Aula / ClickUp | Coordinador de Calidad |
| Métricas | Análisis de defectos, cálculo de indicadores, interpretación de resultados | Aula / práctica | Encargado de Métricas |
| Gestión de Configuración (SCM) | Git, GitHub, ramificación, control de versiones | Aula / práctica | SCM Manager |
| Uso de Herramientas de SQA | ClickUp, Google Drive, plantillas Word, Lucidchart | En aula / en práctica | Equipo Técnico / Coordinador QA |
| Documentación Técnica | Checklists, formularios, bitácoras de revisión | En aula | Revisor de Calidad / Coordinador |
| Validación con el Cliente | Interpretación de requisitos, comunicación clara, feedback estructurado | Taller de rol simulado | Cliente (rol académico) |

# Sección 7 – Informe y Resolución de Problemas de SQA

Las auditorías de calidad se documentan mediante formularios oficiales de evaluación, donde se registran las condiciones observadas, puntos de acción, responsables, acciones correctivas y fechas de cumplimiento. A continuación, se presenta un ejemplo real de incidencia documentada durante el desarrollo del proyecto:

## 7.1 – Incidencias del Proceso de Calidad

**Tabla 11. Formulario de Auditoría de Procesos**

**Formulario de Incidencias del Proceso de Calidad**

Identificador de Seguimiento: INC-SGCPA-004

Auditor Líder: Ing. Fernando Lezcano Fecha del Informe: 20/07/2025

Equipo Auditor: Angel Marquez

Josue Pino

Nombre del Proyecto: Sistema de Gestión de Calidad para Procesos Académicos (SGCPA)

Fecha de la Auditoría: 18/07/2025

Proceso/Procedimiento Auditados: Control de Trazabilidad de Requisitos

Lista de Verificación Utilizada: (Adjuntar) Lista de Auditoría Proceso de Requisitos v1.2

Resultados de la Auditoría: (Marcar uno)

\_\_\_\_\_ Proceso/Procedimiento Aceptable

✓ Proceso/Procedimiento No Aceptable   
 (Sujeto a la finalización satisfactoria de los puntos de acción listados a continuación)

Condiciones observadas: Se detectó que tres requisitos funcionales (RF-03, RF-07, RF-12) no están referenciados en la matriz de trazabilidad ni en el plan de pruebas. Esto representa un riesgo para la cobertura completa de los casos de uso críticos.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Punto de Acción (PA):

Título: Inclusión de Requisitos Faltantes en la Matriz Asignado a: Responsable de Documentación Fecha de entrega:22/07/2025 Fecha de cumplimiento: 24/07/2025

Acción Correctiva: Actualizar la matriz de trazabilidad para incluir los requisitos RF-03, RF-07 y RF-12. Validar con el equipo de pruebas que estos se integren en los casos de prueba correspondientes y sean verificados antes de la entrega final.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Disposición: (Marcar uno) Aprobar  Cancelar  Posponer

✓

Gerente de Proyectos: Edward Camaño Fecha: 25/07/2025

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Cierre del Punto de Acción (PA):

Aprobación SQA: Fernando Lezcano Fecha: 25/07/2025

(Archivar el formulario completado en el Registro de Evaluación de SQA)