Plan de SQA

GRCU Manager

Gestión de Requerimientos y Casos de Uso

4 BYTES

ALVAREZ, Abril

BUTTERFIELD, Nicolas

CARRANZA, Cristian

GAGNA, Martina

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.Un dibujo con letras

El contenido generado por IA puede ser incorrecto.

La Calidad del Software tiene como objetivo brindar la confianza de que el producto final logrará satisfacer los requisitos del cliente.

En el Plan de SQA se reflejan las evaluaciones a realizar, los estándares a aplicar, los productos a realizar, los procedimientos a seguir en la elaboración de los distintos productos y los procedimientos para informar de los defectos detectados a sus responsables y realizar el seguimiento de los mismos hasta su corrección.

Contenido

[**Propósito 7**](#_heading=h.9x9wx67nz4id)

[**Objetivos específicos del Plan de SQA 7**](#_heading=h.t1krp83i5qnz)

[**Referencias 7**](#_heading=h.jgb1l0nm9a2q)

[**Gestión 8**](#_heading=h.ph5wwcdqffz8)

[**Organización 8**](#_heading=h.5u0jurpaam1t)

[**Actividades 9**](#_heading=h.mk2n2mrv3n1a)

[*Actividades consideradas 9*](#_heading=h.t2o02sil6134)

[*Productos clave a considerar 10*](#_heading=h.xr2r2uxmeg4l)

[**Actividades de calidad a realizarse 12**](#_heading=h.jjd2zqvf2v30)

[*Revisión de productos y aplicación de métricas 12*](#_heading=h.2r09oqocdo2h)

[**Revisar cada producto 13**](#_heading=h.jl7cdgxaq60s)

[*Procedimiento de revisión de productos clave 13*](#_heading=h.v3r6nemflbbc)

[**Documentación asociada a cada producto clave 14**](#_heading=h.9nfpl7wkz13p)

[**Revisar el ajuste al proceso 15**](#_heading=h.sempaurk3p1b)

[**Actividad de revisión de productos clave 15**](#_heading=h.z2wcgz1btkms)

[**Documentos de referencia 15**](#_heading=h.k9xq3ddulp2i)

[**Verificación previa 15**](#_heading=h.uxx0xgo7buom)

[*Resultado 15*](#_heading=h.e1quwz2jhvos)

[**Realizar Revisión Técnica Formal (RTF) 16**](#_heading=h.g7a9k9ec6ff3)

[*Objetivo 16*](#_heading=h.is69t0bjh9r1)

[*Proceso de Revisión 16*](#_heading=h.brptt6d4ee3p)

[*Documentación / Entregables 16*](#_heading=h.a7g7htebccg9)

[**Relación con la planificación de SQA 17**](#_heading=h.tc3rr2axixq6)

[**Asegurar que las desviaciones son documentadas 17**](#_heading=h.g3oqsklx78wx)

[**Documentación 19**](#_heading=h.fgx1wgzhgprb)

[*Propósito 19*](#_heading=h.dh6va2r8qyjk)

[*Documentación mínima requerida 19*](#_heading=h.qehisg89i9sp)

[**Especificación de requerimientos del software 20**](#_heading=h.43ol8cdj0o6)

[**Descripción del diseño del software 22**](#_heading=h.mh5cvneul08l)

[*1. Principios del diseño 23*](#_heading=h.elcpjswd9nzs)

[*2. Componentes principales del sistema 23*](#_heading=h.nckdq5kr9ku9)

[*3. Interfaces internas 24*](#_heading=h.jmfcu5snragh)

[*4. Evolución del diseño 24*](#_heading=h.j3g9nvzqc7u)

[**Plan de Verificación & Validación 24**](#_heading=h.76nakanifwtc)

[*1. Verificación 25*](#_heading=h.vjlipf6oblq)

[*2. Validación 25*](#_heading=h.a3cpfz7iolc)

[**Documentación de usuario 26**](#_heading=h.fpywg4icogya)

[*1. Contenido de la documentación 26*](#_heading=h.8f7qtaqkm95n)

[*2. Resultados esperados para el cliente 28*](#_heading=h.8kqga0zcmoxv)

[**Plan de Gestión de configuración 28**](#_heading=h.6gj5cpgigf6f)

[*Propósito 28*](#_heading=h.gtjkb59n1z38)

[*Resumen 28*](#_heading=h.9jsiy1xx43qf)

[**Organización y Responsabilidades 29**](#_heading=h.wt5xexc1qaoe)

[**Herramientas, Entorno e Infraestructura 29**](#_heading=h.q6y8glcqo23d)

[**Forma de Trabajo 30**](#_heading=h.9fil6fuu1iiz)

[**Control de Cambios 30**](#_heading=h.vssb95h1o6v)

[**Reportes y Auditorías 30**](#_heading=h.r20yvn6eqoz4)

[**Otros documentos 31**](#_heading=h.fng10bvg5ies)

[**Estándares, prácticas, convenciones y métricas 32**](#_heading=h.a93q61gxzdio)

[*1. Estándares 32*](#_heading=h.adjs28mp395w)

[*2. Prácticas 32*](#_heading=h.ysbmbvl99gf)

[*3. Convenciones 32*](#_heading=h.d03u93aeaksk)

[*4. Métricas 33*](#_heading=h.7809t53jz7zq)

[*5. Monitoreo y aseguramiento 33*](#_heading=h.jdzro16ml12r)

[*Objetivos 34*](#_heading=h.1ceemob6vzcq)

[**Métricas de proceso 35**](#_heading=h.f4fi7x8q1iu3)

[*Características 35*](#_heading=h.uzpe84ec2lsk)

[*Métricas específicas para este proyecto 35*](#_heading=h.9x781y736d68)

[**Métricas de proyecto 36**](#_heading=h.abe0kdlqo3ab)

[*Características 36*](#_heading=h.3mp2atzhbljw)

[*Métricas específicas para este proyecto 36*](#_heading=h.wdkq3mlweevn)

[**Métricas de producto 37**](#_heading=h.i34ou3z9i99m)

[*Características 37*](#_heading=h.gaka0scysz2j)

[*Métricas específicas para este proyecto 37*](#_heading=h.pvtch0gldk1f)

[**Estándar de documentación 38**](#_heading=h.w1wupy7717ip)

[*Documentación Técnica 39*](#_heading=h.wzcsgdvxtk6u)

[**Documentación de Usuario 39**](#_heading=h.6fpy9bynwm0j)

[**Estándar de verificación y prácticas 40**](#_heading=h.mj6qw7437vp3)

[**Revisiones y auditorías 40**](#_heading=h.n7xn10hd8nqt)

[*Objetivo 40*](#_heading=h.1yf3w1td9zjc)

[*Requerimientos Mínimos 40*](#_heading=h.4zeng3t6l1eo)

[*Tipos de Revisiones y Auditorías 41*](#_heading=h.pi7870x4j1nu)

[**Agenda 42**](#_heading=h.4qfwxpz32e01)

[**Otras revisiones 42**](#_heading=h.x8djwbya20xa)

[**Verificación 43**](#_heading=h.ihzt53kqngdz)

[*Objetivo: 43*](#_heading=h.9lijb7rb8ht0)

[*Métodos a utilizar: 43*](#_heading=h.xumskfjfn6q7)

[*Responsables: 44*](#_heading=h.chcnumxvwf0e)

[*Entregable: 44*](#_heading=h.skaja4wisrlc)

[**Reporte de problemas y acciones correctivas 44**](#_heading=h.3sho8xaonp3q)

[*Objetivo: 44*](#_heading=h.j1fo8niq9mjq)

[*Responsables: 45*](#_heading=h.p8jymyv95kqv)

[*Entregable: 45*](#_heading=h.5dr5jdvbjhze)

[**Herramientas, técnicas y metodologías 45**](#_heading=h.bu2rs1dhfpx7)

[*Objetivo: 45*](#_heading=h.c5ht2i62w28i)

[*Herramientas de Software: 46*](#_heading=h.2n272bt3khzy)

[**Técnicas: 46**](#_heading=h.v9oahsrrjaht)

[*Metodologías: 46*](#_heading=h.at86gmm620dj)

[*Resultados esperados: 47*](#_heading=h.rubzt9uynzjd)

[**Gestión de riesgos 47**](#_heading=h.62l1j1xjfbc)

[*Objetivo: 47*](#_heading=h.s0xwmi6lbhq)

[*Documento de Referencia: 48*](#_heading=h.l649l8gxhtve)

[**Anexos 49**](#_heading=h.p7v69o6lszxa)

[*Formulario de Pedidos y Detección de Cambios 49*](#_heading=h.6aha0aegvdam)

[*Objetivo: 49*](#_heading=h.brssd2bdyqow)

[*Formulario: 49*](#_heading=h.1z6bkshqmvqz)

[*Indicaciones de Uso: 49*](#_heading=h.ksj74mvz979m)

Plan de SQA

# Propósito

El propósito de este plan es definir las actividades necesarias para asegurar la calidad del software a desarrollar. En él se especifican los productos que serán revisados, así como los estándares, normas y reglas que se aplicarán. También se detallan los métodos y procedimientos que se utilizarán para verificar que la elaboración de los productos cumpla con lo establecido en el modelo de ciclo de vida del proyecto. Además, se describen los mecanismos para informar a los responsables sobre los defectos detectados y dar seguimiento a su corrección hasta su resolución.

El software en cuestión consiste en una aplicación web que permite registrar, editar y eliminar requerimientos y casos de uso creados por los alumnos de la cátedra de Laboratorio de Desarrollo de Software. El objetivo principal es garantizar que se realicen las pruebas necesarias para asegurar la calidad del producto final en los proyectos desarrollados.

# Objetivos específicos del Plan de SQA

* Cumplir con los estándares, normas y convenciones establecidas y aceptadas por los integrantes del grupo de desarrollo.
* Promover el compromiso individual de los integrantes del grupo 4Bytes, con el fin de garantizar la calidad del producto.
* Supervisar la elaboración de la documentación asociada y la correcta configuración del software.
* Asegurar que se satisfagan los requerimientos solicitados por los docentes de la cátedra Laboratorio de Desarrollo de Software de la Universidad Nacional de la Patagonia Austral.

# Referencias

[ANSI/IEEE Std 730.1-1989, IEEE Standard for Software Quality Assurance

Documento Plantilla gestión de riesgos

Documento plan de gestión de riesgos

Documento plantilla de revisión de SQA

Documento Informe Final de SQA

Documento plantilla de Revisión técnica formal

SQuaRE, ISO 25000:2005,

Quality management systems – Requirements ISO 9001:2008]

# Gestión

# Organización

Rol Función Responsables

Líder del Proyecto Responsable de la planificación, ejecución y cierre de las actividades del proyecto. Coordina y asigna los recursos necesarios, brinda soporte en las tareas de estimación y definición de actividades incluidas en los planes, y revisa y aprueba los mismos. Cristian Carranza

Administrador de Configuración Encargado del control de versiones y cambios, así como de la coordinación del desarrollo colaborativo. Garantiza que cada producto de trabajo esté disponible en un repositorio que incluya código, documentación y herramientas necesarias, asegurando la integridad y trazabilidad del proyecto. Nicolás Butterfield

Gerentas de Calidad Responsable de garantizar el cumplimiento de los compromisos de calidad del proyecto, conforme a los procesos establecidos. Abril Álvarez, Martina Gagna

Analista Recopila y documenta información relevante para comprender las necesidades del cliente. Prioriza y organiza los requerimientos para que sean claros, completos y viables para el equipo de desarrollo. Abril Álvarez, Martina Gagna

Diseño y Código Responsable del diseño, arquitectura y desarrollo del sistema, asegurando la calidad del código y su cumplimiento con los estándares establecidos. Identifica, implementa y evalúa los factores de calidad aplicables al sistema, verificando que se satisfagan los requisitos técnicos y de negocio. Nicolás Butterfield

Documentador Organiza, mantiene y asegura la calidad de toda la información generada durante el proceso de desarrollo. Garantiza un almacenamiento adecuado para facilitar su recuperación, actualización y coherencia en formato y estructura. Abril Álvarez, Martina Gagna

Administrador de Riesgos Realiza el seguimiento y evaluación continua de los riesgos, proponiendo medidas de mitigación y contingencia para minimizar su impacto. Informa al líder del proyecto cuando un riesgo se convierte en un problema concreto y coordina las acciones necesarias de gestión de riesgos, asegurando que el equipo esté preparado para afrontarlos. Martina Gagna

Pruebas Garantiza que el software cumpla con los requerimientos funcionales establecidos y esté libre de fallas. Planifica, ejecuta y evalúa rigurosamente las pruebas, validando los resultados y verificando la corrección de defectos antes del despliegue final en el ambiente productivo. Abril Álvarez, Martina Gagna, Cristian Carranza

Arquitecto del Sistema Transforma los requisitos definidos por el Analista en una solución técnica coherente y efectiva. Asegura que el desarrollo se ajuste al diseño arquitectónico original, revisándolo y adaptándolo cuando sea necesario para mantener la integridad y alineación con los objetivos del proyecto. Nicolás Butterfield

Todas estas actividades que se realizan durante el desarrollo del proyecto impactan en la calidad del producto final. Las líneas de trabajo que tienen un impacto directo son las siguientes:

• Requerimientos

• Análisis

• Diseño

• Implementación

• Pruebas

# Actividades

Ciclo de vida del software cubierto por el Plan

## Actividades consideradas

● Captura de requerimientos:

○ Reuniones iniciales con el cliente.

○ Revisiones periódicas.

○ Priorización de requerimientos.

○ Aprobación del cliente.

○ Seguimiento mediante solicitudes y control de cambios.

● Análisis y diseño:

○ Aplicación del patrón MVC (o MVT, en caso de migrar a Django).

○ Evaluación de escalabilidad, usabilidad, accesibilidad, compatibilidad y responsabilidad.

○ Revisión mediante RTF, pruebas de usabilidad, auditorías de código y revisiones de arquitectura.

● Implementación:

○ Aplicación de estándares definidos.

○ Revisiones de código.

○ Ejecución de pruebas técnicas.

● Pruebas:

○ Pruebas unitarias.

○ Pruebas de integración.

○ Pruebas funcionales.

○ Pruebas de aceptación de usuario.

○ Pruebas de rendimiento.

○ Pruebas de seguridad.

○ Criterios de calidad: cobertura de pruebas, tasa de errores aceptable, estándares visuales y de usabilidad, y documentación detallada.

## Productos clave a considerar

● Especificaciones de Requerimientos:

○ Descripción funcional.

○ Requerimientos no funcionales.

○ Criterios de aceptación.

○ Priorización.

● Modelo de Caso de Uso:

○ Anexar para facilitar la comprensión de la interacción entre actores y sistema.

○ Verificación de consistencia CU/RF.

○ Identificación clara de actores y casos de uso.

○ Eliminación de ambigüedades.

● Modelo de Diseño:

○ Revisión técnica formal (RTF).

○ Aplicación del modelo MVC (o MVT si se migra a Django).

○ Cumplimiento de principios de encapsulamiento, separación de responsabilidades y coherencia.

○ Verificación de ausencia de inconsistencias en diagramas UML.

○ Asegurar que el código refleje exactamente el diseño.

○ Garantizar seguridad en el control de acceso y manejo de datos.

● Arquitectura del Sistema:

○ Revisión técnica formal (RTF).

○ Verificación de modularidad: componentes bien definidos e independientes, con interacciones claras.

○ Garantía de escalabilidad y mantenibilidad.

○ Posibilidad de modificaciones sin afectar al sistema completo.

○ Cumplimiento de RNF básicos: tolerancia a fallos y seguridad.

○ Alineación con estándares tecnológicos del proyecto para comunicación entre equipos.

○ Flexibilidad para ajustes futuros, reduciendo costos y complejidad.

● Plan de Pruebas:

○ Cobertura completa de cada caso de uso.

○ Alineación con los requerimientos.

○ Priorización de escenarios críticos.

○ Evaluación de la eficacia de las pruebas.

○ Documentación de resultados.

● Plan de Riesgos:

○ Identificación de riesgos.

○ Análisis de impacto y probabilidad.

○ Definición de estrategias de mitigación.

○ Revisión continua.

○ Cobertura completa y validación con lecciones aprendidas.

○ Herramienta utilizada: VESTA Risk Manager.

● Manual de Usuario:

○ Pruebas con usuarios reales (UAT) para obtener retroalimentación directa.

○ Revisión técnica formal (RTF) del manual.

○ Verificación de cobertura de todos los flujos de trabajo y escenarios.

○ Estandarización del lenguaje: sencillo, directo y consistente.

○ Validación de la estructura para asegurar que sea intuitiva, lógica y facilite la navegación y búsqueda de información.

# Actividades de calidad a realizarse

## Revisión de productos y aplicación de métricas

**1. Métricas basadas en SQuaRE (ISO 25000) aplicadas al producto**

● Funcionalidad:

○ Cobertura de requisitos funcionales: mide el porcentaje de requisitos implementados y aprobados.

○ Precisión de resultados: evalúa si los resultados cumplen con lo esperado.

● Fiabilidad:

○ Tasa de fallos: número de fallos ocurridos durante la operación normal en un periodo determinado.

● Usabilidad:

○ Tiempo de aprendizaje: tiempo requerido para que los usuarios aprendan a utilizar la aplicación web.

○ Satisfacción del usuario: medida a través de encuestas sobre la experiencia con la interfaz y el uso general.

● Mantenibilidad:

○ Tasa de modificación: tiempo necesario para modificar el código en respuesta a cambios en los requerimientos.

○ Densidad de defectos: número de defectos por unidad de código (defectos cada 1000 líneas de código).

**2. Métricas basadas en ISO 9001 aplicadas a los procesos de desarrollo**

● Índice de cumplimiento de proceso: mide el grado de alineación de los procesos de desarrollo con los estándares definidos.

● Tiempo de resolución de no conformidades: tiempo promedio para resolver desviaciones que no cumplen con los requisitos.

● Eficacia del proceso de desarrollo: relación entre el número de productos entregados y los problemas identificados y resueltos.

**3. Métricas basadas en IEEE/ANSI 730.1 aplicadas al aseguramiento de la calidad**

● Tasa de defectos en RTF: número de defectos detectados durante las revisiones técnicas formales.

● Cobertura de pruebas: porcentaje de código o requisitos cubiertos por las pruebas.

● Tiempo de corrección de defectos: tiempo promedio necesario para corregir defectos reportados en pruebas o revisiones.

# Revisar cada producto

## Procedimiento de revisión de productos clave

Cada producto definido como clave será revisado siguiendo los siguientes pasos:

1. Verificación de correcciones pendientes: se confirmará que no queden observaciones sin resolver en los informes de revisión anteriores. En caso de existir, deberán incluirse en la nueva revisión.

2. Cumplimiento de estándares: se verificará que los productos cumplan con los estándares definidos para el proyecto.

3. Gestión de desviaciones: se identificarán, documentarán y harán seguimiento de las desviaciones encontradas, asegurando que las correcciones se hayan implementado.

Para facilitar este proceso, se adjuntará como anexo una plantilla en Excel denominada ChecklistProductoClave, que servirá como control de revisiones en cada iteración. La plantilla contendrá diferentes hojas con los chequeos correspondientes a cada entrega. Aunque la Revisión SQA contempla los productos clave, también podrán incluirse otros productos adicionales bajo el sufijo “apoyo”.

Como resultado de este proceso, se generará un Informe de Revisión de SQA, el cual deberá distribuirse a los responsables de cada producto, asegurando que estén al tanto de las desviaciones o discrepancias encontradas.

# Documentación asociada a cada producto clave

Para cada producto clave se generará un documento específico:

● Revisión de SQA – Especificación de Requerimientos de Software

● Revisión de SQA – Modelo de Casos de Uso

● Revisión de SQA – Modelo de Diseño

● Revisión de SQA – Arquitectura del Sistema

● Revisión de SQA – Plan de Pruebas

● Revisión de SQA – Plan de Riesgos

● Revisión de SQA – Manual de Usuario

Cada documento de Revisión de SQA será almacenado en el repositorio de Gestión de Calidad, correspondiente a la fase y iteración en la que se genere.

# Revisar el ajuste al proceso

# Actividad de revisión de productos clave

En esta actividad se revisarán los productos definidos como clave con el fin de verificar su cumplimiento respecto a las actividades y procesos establecidos en el plan del proyecto. El objetivo principal es asegurar la calidad del producto final mediante la aplicación de los estándares y especificaciones definidos a lo largo del ciclo de vida del software.

Durante la revisión se recopilará la información necesaria de cada producto, incluyendo un análisis retrospectivo de aquellos que deberían haberse generado. Esto permitirá establecer criterios de revisión claros y evaluar si el producto cumple con las especificaciones definidas.

# Documentos de referencia

La información necesaria para esta revisión se obtiene de los siguientes documentos:

1. Plan del Proyecto: define el alcance, los objetivos y los procesos del proyecto. (ISO 9001)

2. Plan de la Iteración: describe las actividades y entregables específicos correspondientes a la iteración. (ISO 9001)

3. Plan de Pruebas: establece los métodos y criterios para la validación del producto. (IEEE 730.1)

# Verificación previa

Antes de iniciar, se debe comprobar en los informes de revisión anteriores que todas las desviaciones hayan sido corregidas. En caso contrario, las faltantes deberán incluirse nuevamente para su evaluación.

## Resultado

Como salida de esta actividad se genera el Informe de Revisión de Ajuste al Proceso, el cual debe ser distribuido a los responsables de las actividades. Este informe asegura que estén al tanto de las desviaciones o discrepancias detectadas, guiándose por las normas ISO 9001 e IEEE 730.1.

# Realizar Revisión Técnica Formal (RTF)

## Objetivo

La Revisión Técnica Formal (RTF) tiene como finalidad identificar errores en la lógica, función o implementación del producto software, verificando que se cumplan las especificaciones definidas y que el producto se ajuste a los estándares establecidos.

Este proceso de revisión riguroso está orientado a detectar defectos o desviaciones tempranamente, con énfasis en los productos críticos para el éxito del proyecto.

## Proceso de Revisión

La reunión de RTF debe cumplir con los siguientes lineamientos:

1. Participación: Asistencia obligatoria del responsable de SQA e integrantes del equipo de desarrollo.

2. Convocatoria: Debe ser formal, dirigida a todos los involucrados, e incluir con anticipación el material necesario.

3. Preparación: Los participantes deben elaborar una lista de preguntas y dudas derivadas del estudio previo del producto a revisar.

4. Duración: La reunión no debe superar las dos horas.

## Documentación / Entregables

La RTF debe garantizar la generación y validación de los siguientes entregables:

1. Informe de RTF: Documento principal que registra hallazgos, desviaciones y acuerdos alcanzados durante la revisión.

2. Documentos a revisar:

a) Especificación de Requerimientos de Software.

b) Modelo de Casos de Uso.

c) Modelo de Diseño.

d) Arquitectura del Sistema.

e) Plan de Pruebas.

f) Plan de Riesgos.

g) Manual de Usuario.

# Relación con la planificación de SQA

● Al finalizar cada iteración y en cada fase del proyecto, se realizarán los controles correspondientes a las actividades de SQA.

● Las métricas establecidas en cada control deben ser documentadas y consideradas como insumos para la mejora continua.

● Estas actividades aseguran la trazabilidad entre lo planificado y lo ejecutado, garantizando el cumplimiento de normas ISO 9001 e IEEE 730.1.

# Asegurar que las desviaciones son documentadas

Las desviaciones encontradas en las actividades y en los productos deben ser documentadas y gestionadas según el procedimiento establecido.

Cada responsable debe actualizar sus planes cuando sea necesario, en base a las desviaciones identificadas, garantizando la trazabilidad y la mejora continua.

Tabla de planificación de actividades SQA

Actividad Semana (ejemplo)

Revisión de ajuste al proceso (documentación vs plan del proyecto) Semana 2

Verificación de corrección de desviaciones previas Semana 3

Revisión Técnica Formal (RTF) de Especificación de Requerimientos Semana 4

RTF de Casos de Uso y Modelo de Diseño Semana 6

RTF de Arquitectura del Sistema Semana 8

Revisión de Plan de Pruebas y Plan de Riesgos Semana 9

Control de calidad de entregables de iteración Semana 10

Consolidación de métricas de SQA y elaboración de informe Semana 11

Validación de Manual de Usuario y documentación final Semana 12

Responsables

El Administrador del Proyecto y el Administrador de SQA son responsables de ejecutar los controles de calidad durante la ejecución del proyecto, asegurando que las actividades y productos cumplan con los estándares y requisitos establecidos.

Sus responsabilidades se organizan en el siguiente ciclo:

1. Ejecución de la tarea

● La tarea debe realizarse de acuerdo con los requisitos del plan del proyecto y los estándares definidos.

● Se asigna un responsable: puede ser el equipo de desarrollo en su conjunto o un integrante designado específicamente.

2. Control de revisiones

● Se realiza una revisión de la tarea para verificar su conformidad con los requisitos y estándares.

● Se determina si la tarea es aceptada tal como está o si requiere correcciones.

● Responsable: Administrador de SQA y/o Administrador del Proyecto, según la etapa de revisión.

3. Análisis causal de errores

Cuando se detecta un error o desviación, se aplica un análisis causal que incluye:

1. Análisis del error: comprender cómo y por qué ocurrió.
2. Hipótesis de causa: formular hipótesis sobre la posible causa raíz.
3. Investigación: determinar en qué momento y por qué se produjo el error.
4. Corrección: implementar una solución que resuelva el error.
5. Acción correctiva: establecer medidas preventivas para evitar su recurrencia.

4. Registro y métricas

● Los resultados del análisis causal deben ser documentados para generar métricas de calidad y trazabilidad de problemas.

● Responsable: Administrador de SQA.

5. Reinicio del ciclo

● Se reinicia el ciclo con la ejecución de la tarea corregida y la verificación de que los defectos fueron resueltos.

● Responsable: equipo de desarrollo, supervisado por el Administrador del Proyecto y el Administrador de SQA.

# Documentación

## Propósito

El objetivo del presente apartado es definir la gestión de la documentación clave para la garantía de calidad de software, cubriendo aspectos relacionados con el desarrollo, pruebas, uso y mantenimiento de manera estructurada y conforme a los estándares establecidos.

La documentación constituye un elemento central en las actividades de SQA, ya que permite garantizar la trazabilidad, la transparencia y la coherencia en cada etapa del ciclo de vida del software. Además, asegura que los equipos involucrados dispongan de información clara, actualizada y verificable.

## Documentación mínima requerida

Para asegurar que la implementación del software satisfaga los requerimientos definidos, se considera la siguiente documentación mínima:

1. Especificación de Requerimientos de Software (ERS): detalla los requisitos funcionales y no funcionales que debe cumplir el sistema.

2. Modelo de Casos de Uso: describe la interacción entre los actores y el sistema, facilitando la comprensión de las funcionalidades.

3. Modelo de Diseño: representa la estructura y organización interna del software, asegurando la correcta implementación de los requisitos.

4. Arquitectura del Sistema: define los componentes, módulos y su interacción, garantizando escalabilidad, mantenibilidad y seguridad.

5. Plan de Pruebas: establece los criterios, métodos y cobertura de pruebas necesarias para validar la funcionalidad y calidad del software.

6. Plan de Riesgos: identifica riesgos, analiza su impacto y propone estrategias de mitigación y contingencia.

7. Manual de Usuario: proporciona instrucciones claras para la correcta utilización del software, asegurando usabilidad y comprensión por parte del usuario final.

# Especificación de requerimientos del software

1. Descripción general del software

El sistema es una aplicación web que permite la gestión de requerimientos y casos de uso de proyectos de software.

Su propósito es facilitar la creación, administración y seguimiento de los requerimientos generados por los integrantes de un proyecto, manteniendo un historial de cambios y mostrando las relaciones mediante una matriz de trazabilidad.

El sistema contempla distintos roles de usuario:

● Líder de proyecto: crea proyectos, asigna desarrolladores y supervisa el progreso.

● Desarrolladores: declaran requerimientos, asignan casos de uso y gestionan su progreso.

● Visitantes: pueden visualizar el contenido de los proyectos sin modificarlo.

El acceso al sistema se realiza mediante autenticación con Google, utilizando el framework UARGFlow como base para el desarrollo. Se incluyen funcionalidades de comentarios y documentación adicional asociada a cada requerimiento o caso de uso.

2. Requerimientos funcionales

1. CRUD (Crear, Leer, Actualizar, Eliminar) de requerimientos y casos de uso.

2. Gestión de historial de cambios para cada requerimiento y caso de uso.

3. Visualización de la matriz de trazabilidad mostrando relaciones entre requerimientos y casos de uso.

4. Gestión de roles y permisos según el tipo de usuario (Líder, Desarrollador, Visitante).

5. Asignación de casos de uso a desarrolladores.

6. Añadir comentarios y documentación extra a requerimientos y casos de uso.

7. Acceso mediante Google Sign-In utilizando UARGFlow.

8. Dashboard de seguimiento con Live Traceability para gestionar el progreso del proyecto.

3. Requerimientos no funcionales / Atributos de calidad

3.1 Funcionalidad

● Adecuación a las necesidades: el sistema debe permitir la gestión completa de requerimientos y casos de uso según los flujos de trabajo definidos.

● Precisión de resultados: los cambios y relaciones deben reflejarse correctamente en la matriz de trazabilidad y en el historial.

● Interoperabilidad: el sistema debe integrarse correctamente con Google para autenticación y posibles exportaciones de datos.

● Seguridad de los datos: los datos sensibles de los usuarios y proyectos deben estar protegidos mediante cifrado y control de acceso.

3.2 Confiabilidad

● Madurez: el sistema debe funcionar de manera estable durante el uso diario.

● Tolerancia a fallas: en caso de errores menores, el sistema debe continuar operativo.

● Recuperabilidad: debe ser posible restaurar la información en caso de fallos críticos.

3.3 Usabilidad

● Comprensible: los usuarios deben entender la interfaz sin necesidad de manuales extensos.

● Aprendible: nuevas funcionalidades deben ser aprendidas en menos de 10 minutos para un usuario promedio.

● Operable: todas las acciones del sistema deben ser realizables mediante la interfaz web.

● Atractivo: la interfaz debe ser clara y visualmente agradable.

3.4 Eficiencia

● Comportamiento respecto al tiempo: las operaciones CRUD deben ejecutarse en menos de 2 segundos promedio.

● Utilización de recursos: el sistema debe optimizar el uso de memoria y procesador en entornos estándar de servidor web.

3.5 Mantenibilidad

● Analizable: el código debe estar documentado y organizado para facilitar su comprensión.

● Modificable: el sistema debe permitir cambios sin afectar otras funcionalidades.

● Estable: las modificaciones no deben generar efectos inesperados.

● Verificable: cada módulo debe poder ser testeado de forma independiente.

3.6 Portabilidad

● Adaptable: el sistema debe funcionar en distintos navegadores web modernos.

● Instalable: debe poder desplegarse en servidores estándar de la universidad.

● Co-existencia: no debe interferir con otros sistemas desplegados en el mismo servidor.

4. Procedimientos de verificación

● Pruebas unitarias: verificar cada funcionalidad individualmente (CRUD, autenticación, matriz de trazabilidad).

● Pruebas de integración: comprobar la interacción entre módulos (por ejemplo, historial de cambios y matriz de trazabilidad).

● Pruebas funcionales: validar que los requerimientos funcionales se cumplen correctamente.

● Pruebas de aceptación de usuario (UAT): los usuarios finales (Líder, Desarrolladores, Visitantes) verifican que el sistema cumple sus expectativas.

● Revisiones RTF: revisión formal de documentos y código para asegurar consistencia y cumplimiento de estándares.

# Descripción del diseño del software

El documento de diseño describe cómo será construido el software para satisfacer los requerimientos definidos en la Especificación de Requerimientos de Software (ERS). Este documento abarca tanto componentes como subcomponentes, así como interfaces internas entre ellos, y se desarrolla inicialmente como un diseño Preliminar, que se irá ampliando progresivamente hasta alcanzar un diseño Detallado.

El objetivo es que el cliente obtenga un diseño de software que refleje todos los aspectos acordados previamente, priorizando aquellos de mayor importancia y considerando sus interacciones lógicas dentro del sistema.

## 1. Principios del diseño

El diseño del software debe:

1. Corresponder a los requerimientos a incorporar

○ Cada elemento del diseño contribuye a cumplir uno o más requerimientos funcionales o no funcionales.

○ Todos los requerimientos definidos en la ERS deben estar contemplados en al menos un componente o subcomponente del diseño.

2. Ser consistente con la calidad del producto

○ El diseño debe reflejar los atributos de calidad definidos en la ERS, incluyendo funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad.

○ Debe garantizar que las implementaciones sean verificables, seguras y trazables.

## 2. Componentes principales del sistema

1. Módulo de gestión de proyectos

○ Permite la creación de proyectos, asignación de desarrolladores y gestión de roles.

○ Relacionado con los requerimientos de administración y permisos de usuario.

2. Módulo de gestión de requerimientos y casos de uso

○ CRUD de requerimientos y casos de uso, historial de cambios y asignación de proyectos a desarrolladores.

○ Garantiza trazabilidad entre requerimientos y casos de uso.

3. Módulo de matriz de trazabilidad

○ Visualiza las relaciones entre requerimientos y casos de uso, incluyendo comentarios y documentación adicional.

○ Permite el seguimiento en Live Traceability.

4. Módulo de autenticación y roles

○ Autenticación mediante Google Sign-In utilizando UARGFlow.

○ Control de acceso según roles: Líder de Proyecto, Desarrolladores y Visitantes.

5. Módulo de pruebas y validación

○ Interfaz para ejecutar y documentar pruebas funcionales y de aceptación.

○ Integración con el Plan de Pruebas para verificar cumplimiento de requerimientos.

## 3. Interfaces internas

● Cada módulo se comunica mediante APIs internas claramente definidas.

● Los datos se almacenan de manera centralizada, garantizando integridad y consistencia.

● Las interfaces permiten agregar funcionalidades futuras sin afectar otros módulos.

## 4. Evolución del diseño

● Diseño Preliminar: define la estructura general, módulos principales y relaciones básicas.

● Diseño Detallado: incluye diagramas UML completos, interfaces detalladas, flujos de datos, reglas de negocio y consideraciones de calidad.

● Cada iteración de diseño se revisará mediante RTF y se actualizará según las desviaciones o mejoras detectadas.

# Plan de Verificación & Validación

El Plan de Verificación y Validación tiene como objetivo asegurar que el software desarrollado cumpla con los requerimientos establecidos, que el diseño se implemente correctamente y que el código final satisfaga las expectativas de los usuarios y las normas de calidad definidas. Este plan se aplicará a todos los proyectos asignados a los desarrolladores, quienes definirán los requerimientos y casos de uso correspondientes a cada proyecto.

## 1. Verificación

La verificación garantiza que cada etapa del desarrollo cumple con lo definido en la documentación previa:

1. Verificación de Requerimientos

○ Confirmar que todos los requerimientos declarados por los desarrolladores están aprobados por el Líder de Proyecto o autoridad correspondiente.

○ Asegurarse de que los requerimientos sean claros, completos, consistentes y verificables.

○ Métodos: revisión de documentos, checklist de aprobación, revisiones RTF.

2. Verificación de Diseño

○ Comprobar que cada requerimiento aprobado está contemplado en al menos un componente del diseño del software.

○ Evaluar consistencia y alineación con los atributos de calidad definidos en la ERS.

○ Métodos: revisión de diagramas UML, análisis de trazabilidad diseño-requerimientos, RTF.

3. Verificación de Implementación

○ Confirmar que el diseño detallado se ha implementado correctamente en el código.

○ Asegurar que la arquitectura modular y las interfaces internas se respetan según lo definido en el documento de diseño.

○ Métodos: revisiones de código, pruebas unitarias, cobertura de código, auditorías internas de calidad.

## 2. Validación

La validación asegura que el producto final satisface los requerimientos funcionales y no funcionales expresados en la ERS:

1. Pruebas funcionales

○ Validar que todas las funcionalidades del CRUD, gestión de proyectos, asignación de roles y casos de uso se ejecutan correctamente.

○ Métodos: ejecución de escenarios de prueba según el Plan de Pruebas, revisión de resultados.

2. Pruebas de aceptación de usuario (UAT)

○ Confirmar que los usuarios finales (Líder de Proyecto, Desarrolladores y Visitantes) pueden interactuar con el sistema conforme a sus expectativas.

○ Métodos: sesiones de prueba con usuarios reales, retroalimentación directa, ajustes basados en comentarios.

3. Validación de trazabilidad

○ Verificar que los requerimientos, casos de uso y modificaciones estén reflejados correctamente en la matriz de Live Traceability.

○ Métodos: inspección de la matriz, comparación con historial de cambios, revisiones de comentarios y documentación extra.

3. Procedimientos adicionales

● Todas las desviaciones o defectos detectados durante la verificación y validación se documentarán en el Informe de SQA, asignando responsables y estableciendo un plan de corrección.

● Se realizarán revisiones periódicas después de cada iteración del proyecto para asegurar la conformidad continua del sistema con los estándares de calidad ISO 25000, ISO 9001 y IEEE 730.1.

# Documentación de usuario

La documentación de usuario tiene como objetivo guiar a los usuarios en la correcta utilización del sistema web de gestión de requerimientos y casos de uso, asegurando que puedan interactuar con todas las funcionalidades de forma eficiente y sin errores.

## 1. Contenido de la documentación

La documentación incluirá:

1. Descripción del sistema

○ Propósito del software y roles de usuario (Líder de Proyecto, Desarrollador, Visitante).

○ Funcionalidades principales: CRUD de requerimientos y casos de uso, historial de cambios, matriz de trazabilidad, Live Traceability, comentarios y documentación extra.

2. Datos y entradas de control

○ Requerimientos para iniciar sesión mediante Google Sign-In.

○ Formatos y campos necesarios para la creación de requerimientos y casos de uso.

○ Procedimientos para asignar casos de uso a desarrolladores y proyectos.

3. Secuencia de entradas y uso de la aplicación

○ Flujo de navegación para cada tipo de usuario.

○ Instrucciones paso a paso para crear, modificar y eliminar requerimientos o casos de uso.

○ Uso de la matriz de trazabilidad y seguimiento de cambios.

○ Procedimientos para añadir comentarios y documentación adicional.

4. Opciones y limitaciones del sistema

○ Restricciones de permisos según el rol del usuario.

○ Límites de almacenamiento o número de requerimientos por proyecto (si aplica).

○ Posibles errores y advertencias que el usuario puede encontrar durante la interacción.

5. Manejo de errores y acciones correctivas

○ Descripción de errores comunes (por ejemplo, conflictos al modificar un requerimiento o caso de uso ya asignado).

○ Procedimientos recomendados para corregir o reportar errores.

○ Contacto o guía para soporte técnico interno.

## 2. Resultados esperados para el cliente

Al finalizar el proyecto, el cliente recibirá una documentación de usuario completa, que permitirá:

● Entender y manejar todas las funcionalidades del sistema sin necesidad de asistencia adicional.

● Identificar errores y aplicar las acciones correctivas indicadas.

● Garantizar que la operación del software sea coherente con los requerimientos definidos y que cumpla con los estándares de calidad establecidos en el Plan SQA.

# Plan de Gestión de configuración

## Propósito

El propósito del Plan de Gestión de Configuración es controlar la entrega y el cambio de los elementos del software a lo largo de todo su ciclo de vida, asegurando que:

● Todos los elementos de configuración (documentos, código, diseño, planes) estén correctamente almacenados y versionados.

● Las solicitudes de cambio sean registradas, evaluadas y ejecutadas de manera controlada.

● Se mantenga un historial preciso del estado de cada elemento y de las peticiones de cambio.

## Resumen

La Gestión de Configuración permite:

● Identificar los elementos de un proyecto de desarrollo de software (especificaciones, requerimientos, arquitecturas, código, planes, etc.).

● Controlar las versiones de dichos elementos, asignando responsabilidades a los miembros encargados.

● Generar informes sobre el estado de desarrollo y cambios implementados.

● Reducir errores y mejorar la productividad durante el ciclo de vida del proyecto.

● Facilitar el mantenimiento del sistema, proporcionando información precisa sobre el impacto de cambios evolutivos o correctivos.

# Organización y Responsabilidades

● Administrador de Configuración: Un integrante del grupo será responsable de gestionar los repositorios en GitHub, permisos de acceso, ramas y control de cambios.

● Solicitudes de Cambio: Cualquier miembro del equipo podrá proponer cambios mediante pull requests, que serán revisados y aprobados por el Administrador de Configuración antes de fusionarse en la rama principal.

● Equipo de Desarrollo: Todos los integrantes deberán mantener su repositorio local sincronizado con GitHub, seguir las políticas de ramas (por ejemplo, main para producción, develop para desarrollo, feature/\* para nuevas funcionalidades) y documentar sus cambios correctamente.

# Herramientas, Entorno e Infraestructura

Se utilizarán las siguientes herramientas:

● Git: Sistema de control de versiones distribuido para manejar el código fuente y documentos del proyecto.

● GitHub: Plataforma centralizada para almacenar repositorios, gestionar ramas, pull requests y control de acceso.

● Clientes Git (opcional): Git CLI, GitHub Desktop o extensiones de IDE (como GitHub Copilot/Git Integration en VSCode) para facilitar la gestión de repositorios.

Estas herramientas permiten:

● Registrar todos los cambios en el repositorio, incluyendo autor, fecha y descripción de los commits.

● Mantener un historial completo del proyecto, facilitando la recuperación de versiones anteriores.

● Coordinar el trabajo en equipo de manera eficiente, asegurando que todos los miembros trabajen sobre la versión más reciente y revisada.

● Gestionar revisiones de código mediante pull requests, comentarios y aprobaciones.

# Forma de Trabajo

● Cada miembro del equipo trabajará en su rama de desarrollo (feature/\*) y realizará commits frecuentes para mantener un historial detallado.

● Los cambios serán integrados a la rama principal (main) solo a través de pull requests, revisadas por el Administrador de Configuración y/o el líder del proyecto.

● GitHub servirá como repositorio central para sincronizar los cambios y garantizar que todos los integrantes tengan la última versión.

● Se establecerá un procedimiento de merge y resolución de conflictos para evitar pérdida de información o inconsistencias.

# Control de Cambios

● Solicitud de Cambio: Se realizará mediante pull requests que incluyan:

○ Nombre del elemento de configuración afectado.

○ Nombre del solicitante.

○ Descripción detallada del cambio.

○ Justificación del cambio y prioridad.

○ Estado de revisión (pendiente, aprobado, rechazado).

○ Fecha del commit o merge.

● Todos los cambios deberán ser revisados y aprobados antes de integrarse a la rama principal, asegurando trazabilidad completa.

# Reportes y Auditorías

1. Auditoría Funcional:

○ Verifica que todos los elementos de configuración auditados han completado las pruebas necesarias.

○ Evalúa si los elementos cumplen con los requisitos establecidos.

2. Revisión Formal de Certificación:

○ Certifica que los elementos de configuración funcionan correctamente en su entorno operativo.

○ Permite dar conformidad formal al avance del proyecto y liberar elementos de configuración para uso posterior.

# Otros documentos

Además de la documentación clave y la gestión de configuración, se consideran otros documentos de relevancia para garantizar la calidad del producto:

1. Plan de Desarrollo:

○ Describe las fases de desarrollo, metodologías, herramientas, prácticas y criterios de calidad que se aplicarán durante todo el ciclo de vida del proyecto.

○ Incluye la planificación de tareas, asignación de responsables, estimaciones de tiempo y recursos, y procedimientos para seguimiento de avance.

2. Plan de Proyecto:

○ Documento que especifica el alcance, objetivos, entregables y recursos del proyecto.

○ Define cronogramas, hitos, roles y responsabilidades del equipo, así como procedimientos de control y revisión.

○ Permite evaluar el progreso, identificar desviaciones y aplicar acciones correctivas.

3. Manual de Estándares y Procedimientos:

○ Contiene las normas de codificación, documentación, control de versiones, pruebas y aseguramiento de calidad que el equipo deberá seguir.

○ Proporciona lineamientos claros para garantizar consistencia, trazabilidad y cumplimiento de buenas prácticas durante el desarrollo del software.

4. Documentos adicionales (opcional):

○ Procedimientos de seguridad y acceso al sistema.

○ Guías para la gestión de cambios, riesgos y control de versiones.

○ Plantillas de informes y registros de revisión de SQA.

# Estándares, prácticas, convenciones y métricas

En esta sección se definen los estándares, prácticas, convenciones y métricas que guiarán el aseguramiento de calidad en el proyecto GRCU Manager. Se alinean con los estándares definidos por la cátedra, incluyendo sólo aquellos adaptados específicamente para documentación y la etapa de programación: ISO/IEC 25000 (SQuaRE) para calidad de código y pruebas en programación; ISO 9001 para procesos de documentación; IEEE 730 para verificación en documentación y revisiones de código; CMMI para productividad en programación; TMMi para efectividad en pruebas durante programación; y métricas generales para complejidad y reutilización en código.

## 1. Estándares

El proyecto se regirá por los siguientes estándares:

* ISO/IEC 25000 (SQuaRE): Para medir fiabilidad y calidad del código en programación (e.g., densidad de defectos, cobertura de pruebas). Aplicado en documentación para asegurar trazabilidad de requerimientos.
* ISO 9001: Para control de procesos en documentación (e.g., tiempo de ciclo de desarrollo para producción de docs).
* IEEE 730: Para revisiones y cobertura de pruebas en documentación técnica y de usuario.
* CMMI: Para productividad del equipo en programación (e.g., funcionalidades entregadas por horas-hombre).
* TMMi: Para efectividad en pruebas durante programación (e.g., tasa de defectos en pruebas).
* Métricas generales: Para complejidad de diseño y reutilización de código en programación.

## 2. Prácticas

● Control de versiones: Uso de Git y GitHub para gestión de código y documentación, revisiones mediante pull requests, ramas para desarrollo y producción, y resolución de conflictos.

● Revisiones formales (RTF): Para detectar desviaciones y errores en los productos clave (requerimientos, diseño, arquitectura, pruebas, documentación).

● Pruebas continuas: Unitarias, integración, funcionales, de aceptación de usuario, seguridad y rendimiento.

● Documentación estandarizada: Uso de plantillas uniformes para especificaciones, planes, manuales y reportes de SQA.

Revisiones formales de documentos y código (e.g., RTF) con chequeos basados en IEEE 730.

Monitoreo continuo de métricas en sprints ágiles, con reportes por iteración para docs y código.

## 

## 3. Convenciones

● Nomenclatura de ramas: main para producción, develop para desarrollo, feature/\* para nuevas funcionalidades.

● Formato de commits:

○ Los mensajes de commit deben ser claros y descriptivos, indicando la funcionalidad añadida, la corrección realizada o la mejora implementada. Se seguirá un estándar de comentarios de commits, que incluye:

○ Tipo de cambio: feat (nueva funcionalidad), fix (corrección de error), docs (documentación), refactor (refactorización), test (pruebas), chore (tareas de mantenimiento).

○ Descripción breve: Una línea que resuma el cambio realizado.

● Descripción detallada (opcional): Explicación del cambio, motivo y referencias a issues o tickets si aplica.

● Plantillas de documentación: Checklists, formularios de cambio y reportes de revisión SQA.

● Roles y responsabilidades: Cada miembro conoce su función y los responsables de revisión, aprobación y verificación de cada producto.

Nomenclatura para defectos en docs y código: Clasificados por severidad (crítico, mayor, menor) según CMMI.

Formato de reportes: Tablas estandarizadas con fórmulas y umbrales (e.g., cobertura de pruebas > 85% en programación).

## 4. Métricas

Las métricas permitirán medir y monitorear la calidad del software y de los procesos de desarrollo:

* Detalladas en subsecciones siguientes, adaptadas solo a documentación y programación. Umbrales objetivo: Basados en benchmarks (e.g., densidad de defectos < 3 por KLOC en código).

● Funcionalidad: Cobertura de requisitos, precisión de resultados, interoperabilidad, seguridad de datos.

● Fiabilidad: Tasa de fallos, tolerancia a fallas, recuperabilidad.

● Usabilidad: Tiempo de aprendizaje, satisfacción del usuario, facilidad de uso.

● Eficiencia: Comportamiento frente al tiempo de ejecución y utilización de recursos.

● Mantenibilidad: Capacidad de análisis, modificabilidad, estabilidad y verificabilidad.

● Portabilidad: Adaptabilidad, instalación, co-existencia y reemplazo según sea aplicable.

## 5. Monitoreo y aseguramiento

● Se registrarán métricas en informes de SQA para cada iteración y fase del proyecto.

● Se revisarán métricas en RTF y controles de revisión de producto para identificar desviaciones y áreas de mejora.

● Se ajustarán métricas y estándares conforme a nuevas tendencias, estadísticas del proyecto y resultados de pruebas.

* Frecuencia: Por sprint en programación, por iteración para documentación.

## Objetivos

El programa de métricas del proyecto tiene como propósito principal asegurar la calidad del software y la eficiencia del proceso de desarrollo mediante la medición sistemática de atributos relevantes. Los objetivos específicos son:

1. Documentar las metas del programa de métricas:

○ Definir claramente qué se espera lograr con la medición de métricas antes de asignar recursos al programa.

○ Garantizar que los objetivos estén alineados con la calidad esperada del producto, el cumplimiento de los requerimientos y la eficiencia del equipo de desarrollo.

2. Identificar la información necesaria para alcanzar las metas:

○ Determinar qué métricas deben recolectarse, cómo se obtendrán y qué indicadores proporcionarán información útil.

○ Establecer un marco de referencia que asegure que los datos obtenidos sean confiables, consistentes y comparables a lo largo del proyecto.

3. Utilizar el programa de métricas como herramienta estratégica:

○ Aplicar las métricas no solo para control de calidad, sino también para toma de decisiones, planificación de recursos, priorización de actividades y gestión de riesgos.

○ Permitir retroalimentación continua al equipo de desarrollo para mejorar procesos, detectar desviaciones tempranas y optimizar el rendimiento del proyecto.

4. Implementación de un proceso estructurado en ocho pasos:

○ El proceso guía la creación, recopilación, análisis y utilización de métricas de manera sistemática.

○ Garantiza que cada métrica contribuya al logro de los objetivos estratégicos del proyecto y facilite la mejora continua de la calidad del software.

# Métricas de proceso

Las métricas de proceso se recopilan de todos los proyectos y a lo largo de un período prolongado, con el objetivo de evaluar y mejorar la eficiencia del desarrollo y la calidad de los productos generados. Estas métricas permiten realizar un seguimiento sistemático del control y ejecución del proyecto, así como la efectividad de los procesos implementados.

## Características

● Control y ejecución del proyecto: Permiten verificar si las fases del proyecto se ejecutan según lo planificado y si los recursos se utilizan de manera eficiente.

● Medición de tiempos de las fases: Evaluación del tiempo requerido en cada fase del ciclo de vida para identificar retrasos, cuellos de botella o ineficiencias.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Estándar** | **Métrica de Calidad** | **Cómo se Calcula / Fórmula** | **Para qué sirve (Objetivo)** | **Categoría de Tarea/Actividad** | **Fase Ágil para Comenzar Evaluación** |
| ISO 9001 | Tiempo de ciclo de desarrollo | Fecha entrega - Fecha inicio | Controlar eficiencia en docs y código | Gestión de proyecto | Cada sprint |
| ISO 9001 | Ratio de retrabajo (%) | (Trabajo rehecho / trabajo total) \* 100 | Identificar problemas en procesos | Gestión de calidad | Cada sprint |
| CMMI | Productividad del equipo | Funcionalidades entregadas / horas hombre | Medir eficiencia en programación | Gestión de proyecto | Cada sprint |

## Métricas específicas para este proyecto

1. Costo de remoción de defectos: Mide el esfuerzo económico necesario para corregir errores detectados en diferentes fases del proyecto, ayudando a priorizar la detección temprana de fallas.
2. Cantidad de código reusado: Evalúa la proporción de código reutilizable frente al código desarrollado desde cero, fomentando la eficiencia y reducción de esfuerzo de desarrollo.
3. Distribución de esfuerzo por fase: Analiza cómo se distribuye el trabajo entre las distintas fases (análisis, diseño, implementación, pruebas, documentación), identificando áreas de sobrecarga o subutilización.
4. Efectividad para remover defectos entre fases: Calcula la proporción de defectos detectados y corregidos en cada fase, asegurando que los errores no se propaguen a fases posteriores.
5. Soporte de herramientas para procesos propuestos: Evalúa la efectividad de herramientas utilizadas (GitHub, frameworks de pruebas, UARGFlow, etc.) para facilitar la planificación, control y seguimiento de las actividades del proyecto.
6. Cumplimiento de requisitos (%) (ISO 9001): Requisitos cumplidos / total requisitos \* 100. Objetivo: >95% en documentación de requerimientos por sprint.
7. Porcentaje de requerimientos implementados por unidad de tiempo (ISO 9001): Requerimientos implementados / unidad de tiempo. Monitoreo por sprint en programación para casos de uso.

Estas métricas permiten monitorear la salud del proceso de desarrollo y proporcionan información estratégica para la toma de decisiones y la mejora continua en la gestión de calidad del software.

# Métricas de proyecto

Las métricas de proyecto permiten evaluar el estado general del proyecto, seguir la evolución de los riesgos y medir la eficiencia y productividad del equipo en relación con los objetivos establecidos. Estas métricas proporcionan información clave para la toma de decisiones, la planificación de recursos y la gestión de calidad a lo largo del ciclo de vida del software.

## Características

● Evaluación del estado del proyecto: Permiten determinar si el proyecto avanza según lo planificado, si los entregables cumplen con los requerimientos y si los recursos se están utilizando eficientemente.

● Seguimiento de riesgos: Facilitan identificar, priorizar y mitigar riesgos relacionados con retrasos, cambios en los requerimientos o desviaciones en costos y tiempos.

Estas métricas evalúan avance en documentación y programación (CMMI), detectando desviaciones en sprints ágiles.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Estándar** | **Métrica de Calidad** | **Cómo se Calcula / Fórmula** | **Para qué sirve (Objetivo)** | **Categoría de Tarea/Actividad** | **Fase Ágil para Comenzar Evaluación** |
| CMMI | Ratio entregas a tiempo (%) | (Entregas a tiempo / total entregas) \* 100 | Medir cumplimiento de plazos en código | Gestión de proyecto | Cada sprint |
| CMMI | Cambios sobre requerimientos | Número de cambios durante desarrollo | Medir estabilidad en docs y código | Gestión de requisitos | Cada sprint |

## Métricas específicas para este proyecto

1. Cantidad de puntos de función liberados por unidad de tiempo: Mide la productividad del equipo en función de los requerimientos completados.
2. Costo del desarrollo: Evalúa los recursos económicos invertidos en el desarrollo del proyecto, comparando con el presupuesto inicial.
3. Costo del soporte: Mide el costo asociado a correcciones, mantenimiento y asistencia técnica del software.
4. Horas trabajadas: Controla el esfuerzo humano dedicado en cada fase del proyecto.
5. Tiempo (calendario) transcurrido: Compara el tiempo real frente al tiempo estimado para cada fase o iteración.
6. Distribución del esfuerzo por fase: Permite identificar si se están cumpliendo los planes de asignación de recursos y carga de trabajo.
7. Cambios sobre requerimientos durante el desarrollo: Mide la cantidad y naturaleza de modificaciones solicitadas durante la fase de desarrollo.
8. Cambios sobre requerimientos en operación: Evalúa ajustes realizados luego de la liberación del software para corregir errores o mejorar funcionalidades.
9. Origen de los cambios sobre requerimientos: Permite identificar si los cambios provienen del cliente, del equipo de desarrollo o de incidentes detectados en pruebas.
10. Cronograma vs Estimado: Compara el avance real del proyecto con el planificado para detectar retrasos o adelantos.
11. Costo sobre valor agregado: Relaciona el costo del trabajo realizado con el valor generado, proporcionando una visión de eficiencia económica.
12. Porcentaje de requerimientos implementados por unidad de tiempo: Evalúa la velocidad de cumplimiento de los requerimientos planificados en cada iteración.
13. Cambios sobre requerimientos: <10% del total en programación por sprint, para estabilidad de casos de uso.
14. Ratio entregas a tiempo (%): >90% para entregas de docs y código por sprint.

# Métricas de producto

Las métricas de producto se centran en las características intrínsecas del software y en los artefactos asociados (documentos, modelos, componentes), independientemente del proceso de desarrollo. Su objetivo es evaluar la calidad, tamaño, complejidad y fiabilidad de los productos generados, permitiendo anticipar problemas y optimizar la entrega de software confiable.

## Características

● Se enfocan en los atributos del software, no en cómo fue producido.

● Consideran tanto el código como los documentos, modelos y otros artefactos que conforman el producto.

● Permiten medir tamaño, complejidad, completitud, volatilidad y esfuerzo, proporcionando información clave para la mejora continua.

Miden calidad en programación (ISO/IEC 25000, TMMi, IEEE 730) y documentación (IEEE 730), enfocadas en código y verificación de requerimientos durante sprints.

| **Estándar** | **Métrica de Calidad** | **Cómo se Calcula / Fórmula** | **Para qué sirve (Objetivo)** | **Categoría de Tarea/Actividad** | **Fase Ágil para Comenzar Evaluación** |
| --- | --- | --- | --- | --- | --- |
| ISO/IEC 25000 (SQuaRE) | Densidad de defectos | Defectos encontrados / KLOC | Medir calidad del código | Testing, revisión de código | Cada sprint |
| ISO/IEC 25000 (SQuaRE) | Cobertura de pruebas (%) | (Líneas de código probadas / líneas totales) \* 100 | Medir exhaustividad en programación | Testing | Cada sprint |
| IEEE 730 | Cobertura de pruebas (%) | (Casos de prueba ejecutados / casos planificados) \* 100 | Verificar docs y código | Planificación de QA, Testing | Cada sprint |
| TMMi | Tasa de defectos en pruebas (%) | (Defectos detectados / líneas de código) \* 100 | Evaluar efectividad del testing | Testing | Cada sprint |
| General | Cantidad de código reusado | (Líneas de código reutilizadas / total líneas) \* 100 | Evaluar reutilización en programación | Desarrollo técnico | Cada sprint |

## Métricas específicas para este proyecto

1. Puntos de Caso de Uso (Use Case Points): Evalúa el tamaño del software en función de los casos de uso definidos y su complejidad, reflejando el esfuerzo estimado.
2. Puntos de función (Function Points): Mide la funcionalidad entregada al usuario, considerando entradas, salidas, consultas y archivos internos.
3. Complejidad de diseño (acoplamiento): Mide el grado de dependencia entre módulos, evaluando la mantenibilidad y facilidad de modificación del sistema.
4. Complejidad de código: Analiza la lógica interna del código, incluyendo condiciones, ciclos y anidamientos, para asegurar su claridad y mantenibilidad.
5. Métodos por clase: Evalúa la cohesión y tamaño de las clases en la implementación orientada a objetos.
6. Profundidad y ancho de jerarquías: Mide la estructura de herencia y composición en el software, identificando posibles problemas de diseño.
7. Cantidad de objetos y relaciones de colaboración: Evalúa la complejidad y densidad de interacciones entre objetos en el sistema.
8. Volatilidad de componentes: Mide la frecuencia con la que los componentes cambian durante el desarrollo y mantenimiento.
9. Complejidad de despliegue: Evalúa el esfuerzo y riesgo asociados a la instalación y configuración del sistema en diferentes entornos.
10. Densidad de defectos: Número de defectos detectados por unidad de código o por módulo, para evaluar la calidad del software.
11. Tipo y origen de defectos: Clasificación de los defectos según su naturaleza (funcional, de diseño, de integración) y origen (desarrollo, pruebas, usuario).
12. Cantidad de problemas reportados: Registra el total de incidencias detectadas durante pruebas o uso real.
13. Tiempo transcurrido entre fallas: Evalúa la confiabilidad del software y la frecuencia de errores en operación.
14. Tiempo esperado para la siguiente falla: Estimación de confiabilidad basada en métricas históricas de defectos y fallas.
15. Tiempo requerido para reparar: Mide la eficiencia del equipo en la corrección de errores reportados.
16. SLOC (Source Lines of Code): Mide el tamaño del código fuente como indicador de complejidad y esfuerzo de mantenimiento.
17. Facilidad de aprendizaje de uso: Evalúa la usabilidad del software mediante pruebas con usuarios, considerando la curva de aprendizaje y satisfacción de los mismos.
18. Densidad de defectos: <3 por KLOC en código de gestión de requerimientos por sprint.
19. Cobertura de pruebas (%): >85% para verificación de casos de uso en docs y código por sprint.
20. Complejidad de código: Complejidad ciclomática <10 por módulo en programación por sprint.

# Estándar de documentación

Para garantizar la uniformidad y calidad de la documentación generada durante el proyecto, se establecen dos estándares:

1. Estándar de documentación técnica
2. Estándar de documentación de usuario
3. Se aplican estándares IEEE 730 para revisiones de docs, asegurando cobertura completa de requerimientos en GRCU Manager por sprint.

## Documentación Técnica

La documentación técnica debe:

● Ser adecuada para que un grupo independiente del equipo de desarrollo pueda realizar mantenimiento del producto.

● Incluir:

○ Código fuente relevante.

○ Modelos de Casos de Uso.

○ Objetos de diseño y diagramas asociados.

● Seguir las plantillas definidas para todos los entregables, asegurando consistencia en:

○ Encabezado y pie de página.

○ Fuente y tamaño para texto normal.

○ Fuente y tamaño para títulos.

○ Datos mínimos a incluir: fecha, versión y responsables del documento.

* Incluye verificación de código según IEEE 730 durante programación por sprint.

# Documentación de Usuario

La documentación de usuario debe:

* Ser clara, concisa y orientada a facilitar la operación y comprensión del software.
* Incluir procedimientos de uso, explicación de entradas/salidas y guía para resolución de errores comunes.
* Seguir la plantilla establecida para asegurar consistencia en formato, estilo y contenido.
* Guías de usuario y soporte para clientes, con encuestas ISO 9001 para satisfacción. Alineado con TMMi para pruebas de usabilidad en cada sprint.

Nota: Todos los documentos generados deberán cumplir con el Estándar de Documentación definido en Doc01-Estándar de Documentación, que será la referencia oficial para la creación y revisión de entregables dentro del proyecto.

# Estándar de verificación y prácticas

Para garantizar la calidad y confiabilidad del software, se aplican las prácticas definidas en el Plan de Verificación y Validación (V&V) del proyecto.

Estas prácticas incluyen:

● Verificación de que los requerimientos están correctamente implementados en el diseño y en el código.

● Validación de que el software cumple con los requerimientos funcionales y no funcionales cuando es ejecutado.

● Registro de hallazgos, desviaciones y seguimiento hasta su resolución.

● Evaluación sistemática mediante revisiones formales, pruebas unitarias, pruebas de integración y pruebas de aceptación por el usuario.

**Revisiones y auditorías**

* **Objetivo**: Asegurar cumplimiento de estándares en docs y código por sprint.
* **Requerimientos Mínimos** : Criterios IEEE 730 para docs (cobertura 100%) y TMMi para código por sprint.
* **Tipos de Revisiones y Auditorías**: Revisiones técnicas en programación, auditorías de docs por sprint.
* **Agenda**: Cronograma por sprint.
* **Otras revisiones**: Ad hoc para cambios en docs por sprint.

**Verificación**

* **Objetivo**: Verificar docs y código con IEEE 730 por sprint.
* **Métodos a utilizar** Placeholder: Inspecciones con SonarQube para código, revisiones manuales para docs por sprint.
* **Responsables**: Equipo SQA por sprint.

**Reporte de problemas y acciones correctivas**

* **Objetivo**: Resolver defectos en docs y código por sprint.
* **Responsables**: Equipo de programación por sprint.
* **Entregable**: Informes con CMMI por sprint.

#### Herramientas, técnicas y metodologías

* **Objetivo** Definir herramientas para documentación y programación, asegurando trazabilidad con estándares adaptados en un entorno ágil.
* **Herramientas de Software**
  + Google Forms: Encuestas para docs de usuario (ISO 9001) por sprint.
  + Git: Versiones en programación por sprint.
* **Técnicas**
  + Análisis estático y revisiones pares (IEEE 730) para docs por sprint.
* **Metodologías**
  + Metodología Ágil (Scrum) para iteraciones en sprints; mejora continua con CMMI y TMMi por sprint.
* **Resultados esperados**
  + 85% cobertura en docs y código por sprint.

#### Gestión de riesgos

* **Objetivo**: Mitigar riesgos en docs y programación con Vesta Risk Manager por sprint.
* **Documento de Referencia** : Plan de Riesgos. Vesta Risk Manager

#### Anexos

**Objetivo de Referencia**

* Proporcionar formularios para tracking en docs y programación por sprint.

**Anexo 1: Formulario de Pedidos y Detección de Cambios**

**Objetivo**

* Registrar cambios en docs y código, midiendo estabilidad (CMMI) por sprint.

**Formulario**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID Cambio** | **Descripción del Cambio** | **Impacto en Requerimientos (%)** | **Fecha Solicitud** | **Responsable** | **Estado** | **Métrica Asociada** |
| CAM-001 | Agregar filtro en casos de uso | Reducción 10% en docs | 2025-09-17 | Alvarez | Pendiente | Cambios <10% |

**Anexo 2: Formulario de Reporte de Métricas Mensuales**

**Objetivo**

Monitorear métricas para docs y programación (ISO 9001, TMMi) por sprint.

**Formulario**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Mes/Año** | **Categoría** | **Métrica** | **Valor** | **Umbral** | **Observaciones** | **Responsable** |
| Sep-2025 | Documentación | Cumplimiento (%) | 95% | >95% | Docs de requerimientos OK | Butterfield |
| Sep-2025 | Programación | Densidad Defectos | 2.5 | <3 | Bajo en código de casos | Gagna |

**Indicaciones de Uso**

* Completar por sprint con SonarQube/Jira.

Nota: Todo el equipo deberá seguir estas prácticas para asegurar consistencia, trazabilidad y control de calidad durante el ciclo de vida del software.

# Revisiones y auditorías

## Objetivo

Definir las revisiones y auditorías técnicas y de gestión que se realizarán a lo largo del desarrollo del proyecto, así como especificar cómo se llevarán a cabo estas actividades. El propósito es garantizar la calidad del software, la coherencia con los requerimientos y la correcta ejecución de los procesos definidos en el plan.

## Requerimientos Mínimos

Se especifican las revisiones y auditorías que deben realizarse como mínimo, asegurando que cada fase del proyecto se cumpla correctamente y los entregables sean consistentes con los objetivos del proyecto.

## 

## Tipos de Revisiones y Auditorías

1. Revisión de Requerimientos

○ Objetivo: Asegurar que los requerimientos del Cliente están correctamente documentados, completos y claros.

2. Revisión de Diseño Preliminar

○ Objetivo: Verificar la consistencia y suficiencia técnica del diseño preliminar del software.

3. Revisión de Diseño Crítico

○ Objetivo: Confirmar que el diseño detallado es consistente con la especificación de requerimientos y que permite la correcta implementación del software.

4. Auditoría Funcional

○ Objetivo: Comprobar que todos los requerimientos funcionales han sido implementados correctamente antes de la liberación del software.

5. Auditoría Física

○ Objetivo: Verificar que el software y la documentación asociada son consistentes, completos y aptos para la liberación.

6. Auditorías Internas al Proceso

○ Objetivo: Evaluar la consistencia entre:

■ Código vs diseño documentado

■ Especificaciones de interfaz vs implementación

■ Requerimientos funcionales vs descripciones de testeo

7. Revisiones de Gestión

○ Objetivo: Asegurar la ejecución de todas las actividades del Plan SQA.

○ Se realizarán periódicamente por una persona ajena al grupo de desarrollo.

8. Revisión del Plan de Gestión de Configuración

○ Objetivo: Asegurar la consistencia y completitud de los métodos especificados en el plan de gestión de configuración.

9. Revisión Post Mortem

○ Objetivo: Al concluir el proyecto, evaluar las actividades de desarrollo implementadas y generar recomendaciones para futuros proyectos.

# Otras revisiones

Revisión de Documentación de Usuario

● Objetivo: Garantizar que la documentación de usuario sea completa, clara y fácil de aplicar por los distintos roles del sistema (Líder de Proyecto, Desarrolladores, Visitantes).

● Alcance de la revisión:

1. Completitud: Verificar que todos los módulos, funcionalidades y flujos del sistema estén documentados.

2. Claridad: Evaluar que las instrucciones sean comprensibles y estén redactadas en un lenguaje directo y uniforme.

3. Aplicación de uso: Comprobar que los ejemplos, capturas de pantalla y procedimientos permitan al usuario ejecutar correctamente las tareas del sistema, incluyendo la gestión de proyectos, requerimientos, casos de uso y trazabilidad en vivo.

● Responsable de la revisión: Administrador de SQA y Documentador.

● Entrega: Informe de revisión de documentación de usuario, que incluya observaciones y sugerencias de mejora.

# Verificación

## Objetivo:

Asegurar que todos los elementos del software cumplan con los requerimientos definidos, incluyendo aquellos aspectos que no fueron contemplados explícitamente en el Plan de Verificación y Validación inicial.

Alcance de la verificación:

1. Revisar módulos y componentes adicionales generados durante el desarrollo de los proyectos asignados a los desarrolladores.

2. Evaluar que todos los requerimientos definidos por el equipo y aprobados por el Líder de Proyecto estén correctamente implementados.

3. Comprobar la consistencia entre los documentos de requerimientos, casos de uso, diseño, y el código final.

## Métodos a utilizar:

● Revisiones de código (Code Reviews) por pares y por el Administrador de SQA.

● Pruebas unitarias y de integración para verificar que cada módulo cumpla con su función definida.

● Comparación con documentos de diseño y especificación de requerimientos para asegurar trazabilidad completa.

● Inspección de la documentación generada, incluyendo comentarios en commits de Git/GitHub y documentación de usuario.

## 

## Responsables:

● Administrador de SQA: supervisión de la verificación global y generación del informe.

● Equipo de desarrollo: ejecución de pruebas y corrección de desviaciones detectadas.

## Entregable:

Informe de Verificación, incluyendo hallazgos, desviaciones, y acciones correctivas recomendadas.

# Reporte de problemas y acciones correctivas

## Objetivo:

Garantizar que todos los problemas detectados durante el desarrollo, pruebas o verificación del software sean reportados, documentados, gestionados y resueltos de manera ordenada, asegurando que no se repitan y que se mantenga la calidad del producto final.

Prácticas y procedimientos:

1. Detección y reporte de problemas:

○ Cualquier miembro del equipo que identifique un problema (error de código, inconsistencia en documentación, desviación de requerimientos, fallo en pruebas) debe registrarlo inmediatamente en el sistema de seguimiento de incidencias (por ejemplo, GitHub Issues).

○ Cada problema registrado debe incluir: descripción detallada, módulo afectado, pasos para reproducirlo, prioridad, y responsable asignado.

2. Seguimiento de problemas:

○ El Administrador de SQA revisará periódicamente todos los problemas reportados.

○ Se realizará un seguimiento del estado de cada incidencia: pendiente, en progreso, corregido, verificado.

○ Se documentarán las decisiones tomadas y las correcciones aplicadas.

3. Resolución y acciones correctivas:

○ Los problemas críticos serán asignados inmediatamente al responsable de desarrollo correspondiente para su resolución.

○ Se implementarán acciones correctivas para prevenir la recurrencia de problemas similares, incluyendo ajustes en procedimientos, documentación y revisiones de código.

○ Se actualizarán los planes y documentos asociados según sea necesario para reflejar los cambios realizados.

## Responsables:

● Administrador de SQA: supervisión del registro, seguimiento y verificación de la resolución de problemas.

● Equipo de desarrollo: implementación de correcciones y acciones correctivas.

● Líder de proyecto: validación de las soluciones y aprobación de los cambios realizados.

## Entregable:

● Informe de problemas y acciones correctivas, incluyendo descripción de cada incidencia, estado, acciones tomadas y medidas preventivas implementadas.

# Herramientas, técnicas y metodologías

## Objetivo:

Proveer soporte efectivo a las actividades de aseguramiento de calidad del software, facilitando la verificación, validación, control de cambios, seguimiento de incidencias y documentación de los productos del proyecto.

## 

## Herramientas de Software:

1. Git y GitHub: Control de versiones, gestión de ramas, seguimiento de cambios y colaboración entre los integrantes del equipo 4Bytes.

2. GitHub Issues: Registro y seguimiento de problemas, asignación de responsables y trazabilidad de acciones correctivas.

3. UARGFlow: Framework base para el desarrollo del sistema web, manejo de autenticación con cuentas de Google, control de roles y flujo de proyectos.

4. Trello: Organización de tareas, planificación de iteraciones y seguimiento del progreso de actividades.

5. Herramientas de documentación colaborativa (Google Docs): Para mantener documentación técnica y de usuario actualizada y centralizada.

# Técnicas:

1. Revisión Técnica Formal (RTF): Para detectar defectos y desviaciones en productos críticos durante el desarrollo.

2. Revisión de Gestión y Auditorías: Control periódico de la ejecución de actividades y cumplimiento de planes.

3. Checklist de productos clave: Listado de verificación para asegurar cumplimiento de estándares y requisitos antes de la entrega.

4. Pruebas unitarias y de integración: Evaluación sistemática de la funcionalidad y consistencia del software.

5. Análisis causal de errores: Para identificar la causa raíz de problemas y aplicar acciones correctivas.

## Metodologías:

1. Ciclo de vida iterativo e incremental: Permite planificar, desarrollar, revisar y mejorar productos de manera progresiva.

2. Gestión basada en roles: Cada miembro del equipo tiene responsabilidades definidas (líder de proyecto, desarrollador, administrador SQA, visitante) para asegurar trazabilidad y control de calidad.

3. Trazabilidad activa (Live Traceability): Seguimiento continuo de la relación entre requerimientos, casos de uso y tareas implementadas, asegurando que todas las modificaciones se registren y controlen.

## Resultados esperados:

● Reducción de defectos y errores críticos en los productos.

● Cumplimiento de estándares de calidad definidos.

● Seguimiento efectivo de incidencias y acciones correctivas.

● Documentación clara y centralizada que facilite mantenimiento y evolución futura del sistema.

# Gestión de riesgos

## Objetivo:

Identificar, monitorear y controlar los riesgos asociados al desarrollo del sistema web de gestión de requerimientos y casos de uso, asegurando la calidad y continuidad del proyecto.

Métodos y Procedimientos:

1. Identificación de riesgos:

○ Riesgos técnicos: errores en la implementación, fallas en la integración del sistema, problemas con el framework UARGFlow o dependencias externas.

○ Riesgos de planificación: retrasos en la entrega de requerimientos o asignación de tareas incorrecta.

○ Riesgos de calidad: defectos en el software que afecten funcionalidad, trazabilidad o seguridad de los datos.

2. Evaluación de impacto y probabilidad:

○ Cada riesgo se clasificará según su impacto (alto, medio, bajo) y probabilidad de ocurrencia.

○ Se priorizarán los riesgos de mayor impacto y probabilidad.

3. Estrategias de mitigación:

○ Implementar revisiones técnicas periódicas y pruebas unitarias e integrales.

○ Revisar continuamente los requerimientos y casos de uso asignados a los desarrolladores.

○ Documentar y comunicar cualquier cambio en los requisitos o en la planificación del proyecto.

4. Monitoreo y seguimiento:

○ Se utilizará la herramienta Vesta RiskManager para registrar, administrar y dar seguimiento a todos los riesgos del proyecto.

○ Esta herramienta permite centralizar información sobre los riesgos, actualizar su estado, asignar responsables y mantener un historial de acciones correctivas implementadas.

○ Las revisiones periódicas del Documento de Gestión de Riesgos se complementarán con la información registrada en Vesta RiskManager para evaluar la eficacia de las medidas aplicadas.

5. Plan de contingencia:

○ Para cada riesgo crítico identificado, se definirá un plan de acción alternativo que permita minimizar impactos en el proyecto, asegurando la continuidad del desarrollo y el cumplimiento de los objetivos de calidad.

## Documento de Referencia:

Todos los riesgos identificados, junto con su estrategia de mitigación y plan de contingencia, se documentarán en el Documento de Gestión de Riesgos, integrando además la información registrada en Vesta RiskManager como fuente oficial de seguimiento y control.

# Anexos

## Formulario de Pedidos y Detección de Cambios

## Objetivo:

Registrar formalmente cualquier solicitud de cambio sobre los elementos de configuración del proyecto, permitiendo su seguimiento y control durante el ciclo de vida del software.

## Formulario:

● Fecha de Petición: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

● Nombre y Versión del Elemento de Configuración: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

● Nombre del Solicitante: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

● Necesidad del Cambio: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

● Descripción Detallada del Cambio Pedido: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

● Prioridad: Alta / Media / Baja

● Estado: Pendiente / En Proceso / Aprobado / Rechazado

● Fecha de Implementación del Cambio: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

● Identificador de la Nueva Versión: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

● Elementos Afectados por este Cambio: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 

## 

## Indicaciones de Uso:

● Cada cambio solicitado debe ser completado con toda la información requerida antes de ser evaluado.

● El formulario será revisado y aprobado por el Administrador de Configuración y el Administrador de SQA.

● Una vez aprobado, el cambio será registrado en el sistema de control de versiones (Git/GitHub) y en la documentación correspondiente.