# Actividad 2 - Calidad de Software

# Métricas a Evaluar

## Modularidad del Código

### Descripción

El sistema debe estar dividido en módulos o componentes independientes. Deben ser fácilmente extensibles o modificables para implementar cambios en los requerimientos y nuevas funcionalidades.

### Forma que vamos a usar para medir y valores aceptados

**Analizando los módulos del sistema podemos calcular a los módulos en 3 categorías:**

- Altamente Modularizado: 1 Funcionalidad principal sin dependencias.
- Medianamente Modularizado: 2-3 Funcionalidades estrechamente relacionadas.
- No Modularizado: 4 o más Funcionalidades.

**Dependiendo la categoría asignamos un puntaje a cada categoría:**

- Altamente Modularizado → 1.0 Puntos.
- Medianamente Modularizado → 0.7 Puntos.
- No Modularizado → 0.4 Puntos.

**La forma de medir la modularidad será:**
$Cantidad\ de\ puntos\ totales\ /\ Cantidad\ de\ Módulos\ identificados$. A partir del resultado de esa fórmula, establecemos los valores para la escala de la métrica según el tamaño el proyecto a analizar:

1. **Proyecto Chico:** Un proyecto de pocos módulos debe estar bien enfocado y con modularidad bien realizada.
    a. Satisfactorio
        i. Excede los requerimientos: 1.0
        ii. Rango aceptado: Mayor o igual a 0,90
        iii. Mínimamente Aceptable: Mayor o igual a 0,85
    b. Insatisfactorio
        i. Inaceptable: Menor a 0,85
2. **Proyecto Mediano:** Un proyecto con un número mayor de funcionalidades puede tener módulos más o menos modularizados pero es una prioridad mantener la modularidad.
    a. Satisfactorio
        i. Excede los requerimientos: Mayor o igual a 0,90
        ii. Rango aceptado: Mayor o igual a 0,85
        iii. Mínimamente Aceptable: Mayor o igual a 0.75

    b. Insatisfactorio
        i. Inaceptable: Menor a 0.75
3. **Proyecto Grande:** La escala de un proyecto de mayor tamaño hace que aparezcan módulos más densos o con lógica que atraviesa más de un módulo.
    a. Satisfactorio
        i. Excede los requerimientos: Mayor o igual a 0.85
        ii. Rango aceptado: Mayor o igual a 0.75
        iii. Mínimamente Aceptable: Mayor o igual a 0.65
    b. Insatisfactorio
        i. Inaceptable: Menor a 0.65

# Documentación

## Descripción

El producto debe contar con todos sus requerimientos funcionales completamente documentados. Todo flujo principal de la aplicación debe estar documentado de extremo a extremo, contemplando casos de éxito y de error.

## Forma que vamos a usar para medir y valores aceptados

Para analizar la documentación del producto, se va a tener en cuenta los siguientes criterios:

- **Completitud:** ¿La documentación cubre todos los aspectos necesarios del sistema?
- **Exactitud**: ¿La información reflejada es correcta y está alineada con el software?
- **Claridad**: ¿El contenido es comprensible para su audiencia objetivo?

La escala de cada categoría es:

- **Completitud:** $Funcionalidades\ documentadas\ /\ Funcionalidades\ totales.$
- **Exactitud:** $Funcionalidades\ correctas\ /\ Funcionalidades\ documentadas.$
- **Claridad:**
    - 0: Ininteligible.
    - 0.2: Pobre claridad.
    - 0.4: Baja claridad.
    - 0.6: Mediana claridad.
    - 0.8: Buena claridad.
    - 1: Alta claridad.

La calidad final se calcula con la siguiente fórmula:
$(Completitud\ *\ 0.4)\ +\ (Exactitud\ *\ 0.4)\ +\ (Claridad\ *\ 0.2)$. A partir del resultado, establecemos los valores para la escala de la métrica según el tamaño el proyecto a analizar:

1. **Proyecto Chico:** La escala de un proyecto de menor tamaño no requiere de una documentación extensa y detallada.
    a. Satisfactorio
        i. Excede los requerimientos: Mayor o igual a 0.85

        ii. Rango aceptado: Mayor o igual a 0.75
        iii. Mínimamente Aceptable: Mayor o igual a 0.65
    b. Insatisfactorio
        i. Inaceptable: Menor a 0.65
2. **Proyecto Mediano:** Un proyecto con un tamaño mediano implica mayores funcionalidades, por lo que requiere mayor documentación.
    a. Satisfactorio
        i. Excede los requerimientos: Mayor o igual a 0,90
        ii. Rango aceptado: Mayor o igual a 0,85
        iii. Mínimamente Aceptable: Mayor o igual a 0.75
    b. Insatisfactorio
        i. Inaceptable: Menor a 0.75
3. **Proyecto Grande:** Un proyecto grande requiere de mucha documentación para llevar las funcionalidades al detalle.
    a. Satisfactorio
        i. Excede los requerimientos: 1.0
        ii. Rango aceptado: Mayor o igual a 0,90
        iii. Mínimamente Aceptable: Mayor o igual a 0,85
    b. Insatisfactorio
        i. Inaceptable: Menor a 0,85

# Gestión de Logs

## Descripción

El sistema debe generar logs estructurados y detallados para eventos clave del sistema. Deben permitir el análisis con herramientas de monitoreo y control automatizados.

## Forma que vamos a usar para medir y valores aceptados

Para asegurar una gestión de logs efectiva y que cumpla con los requisitos de calidad, se deben considerar los siguientes aspectos:

- **Información Detallada y Contextual:** Cada entrada de log debe proporcionar suficiente contexto para entender qué ocurrió, cuándo y dónde. **Esto incluye**:
    - **Timestamp:** Marca de tiempo precisa del evento.
    - **Nivel de Severidad:** Clasificación del evento
    - **Módulo/Componente:** Identificación clara del módulo o componente
    - **Información Específica del Evento:** Detalles relevantes sobre la acción que se realizó, los datos involucrados (sin incluir información sensible a menos que sea estrictamente necesario y se gestione de forma segura), y el resultado.
    - **Información del Usuario (si aplica):** Identificador del usuario.
    - **Información del Host/Instancia:** Identificación del servidor o instancia donde ocurrió el evento.

- **Estructura Consistente:** Los logs deben seguir un formato consistente). Esto facilita el parsing y el análisis automatizado por herramientas de monitoreo. Un ejemplo en formato JSON podría ser:
- **Niveles de Log Adecuados:** Definir y utilizar correctamente los niveles de severidad permite filtrar la información según las necesidades (e.g., solo mostrar errores en producción, pero incluir información de debug durante el desarrollo). Los niveles comunes incluyen (**DEBUG, INFO, WARNING, ERROR**, **CRITICAL)**
- **Almacenamiento y Retención:** Definir políticas de almacenamiento y retención de logs. Esto incluye decidir dónde se almacenarán los logs
- **Rotación de Logs:** Implementar mecanismos de rotación de logs para evitar que los archivos crezcan indefinidamente y consuman demasiado espacio en disco.
  **Seguridad de los Logs:** Proteger la información sensible que pueda contener los logs mediante controles de acceso adecuados y, si es necesario, técnicas de anonimización o enmascaramiento.
- **Alertas y Notificaciones:** Definir reglas para generar alertas automáticas basadas en patrones específicos en los logs

**Para medir la calidad de los logs hacemos:**

- **Cantidad de puntos totales / Cantidad de aspectos del log analizados:**
- **Satisfactorio**
  - **Buena calidad de logs:** entre 0,8 y 1,0
  - **Aceptable calidad de logs:** entre 0,4 y 0,8
- **Insatisfactorio**
  - **Mala calidad de logs:** entre 0,0 y 0,4
- **Presencia de Timestamps**
  - ¿Cada log generado incluye una marca de tiempo precisa?
  - **aceptable (1,0)** - timestamp preciso
  - **intermedio (0,5)** - timestamp poco preciso, podria tener mas datos
  - **incompleto (0.25)** - timestamp impreciso, falta información
- **Formato Estructurado**
  - ¿Los logs siguen un formato consistente (e.g., JSON)?
  - **aceptable (1,0)** - define y sigue por convención un formato específico de log
  - **intermedio (0,5)** - define pero no siempre sigue un formato específico de log
  - **incompleto (0,25)** - no define o nunca sigue un formato específico de log
- **Niveles de Severidad Correctos**
  - ¿Se utilizan los niveles de log de manera apropiada para clasificar los eventos?
  - **aceptable (1,0)** - tiene suficientes niveles y es específico
  - **intermedio (0,5)** - podría tener mas niveles y ser mas especifico
  - **incompleto (0,25)** - faltan niveles mínimos para ser lo suficientemente específico
- **Información Esencial**
  - ¿Los logs contienen información relevante para el diagnóstico y seguimiento de eventos clave?
  - **aceptable (1,0)** - tiene suficientes información sobre el evento
  - **intermedio (0,5)** - tiene poca información sobre el evento

    - **incompleto (0,25)** - nula información del evento específico
- **Identificación de Componentes**
    - ¿Se identifica claramente el módulo o componente que generó el log?
    - **aceptable (1,0)** - detalla el componente específico que emite el log
    - **intermedio (0,5)** - nombra el componente específico que emite el log
    - **incompleto (0,25)** - no se sabe que componente emite el log
- **Almacenamiento Configurado**
    - ¿Existe una configuración clara para el almacenamiento de logs (ubicación, tipo)?
    - **aceptable (1,0)** - define claramente la configuración del log con una ubicación específica
    - **intermedio (0,5)** - configura el log con la ubicación por defecto
    - **incompleto (0,0)** - no se configura donde se guardan los logs
- **Seguridad (si aplica)**
    - ¿Se han implementado medidas de seguridad para proteger la información sensible en los logs?
    - **aceptable (1,0)** - se implementan medidas de seguridad como permisos sobre los logs
    - **incompleto (0,0)** - no se implementan medidas de seguridad
- **Alertas Definidas (si aplica)**
    - ¿Se han definido reglas de alerta basadas en los logs para eventos críticos?
    - **aceptable (1,0)** - se definen alertas y se activan correctamente
    - **intermedio (0,5)** - se definen alertas pero existen casos donde no se activan
    - **incompleto (0,5)** - no se definen o nunca se activan las alertas

# Manejo de Excepciones

## Descripción

El sistema debe capturar y manejar correctamente las excepciones sin comprometer la estabilidad. Todos los errores del sistema deben estar capturados para generar respuestas controladas al usuario.

## Forma que vamos a usar para medir y valores aceptados

En el marco de una evaluación integral de la calidad del producto software, el manejo de excepciones constituye un pilar esencial para garantizar la fiabilidad y estabilidad del sistema. De acuerdo con las directivas establecidas en este trabajo, *el sistema debe capturar y manejar correctamente las excepciones sin comprometer su estabilidad, asegurando que todos los errores estén contemplados y se traduzcan en respuestas controladas al usuario*. Para operacionalizar esta evaluación, se ha definido una tabla con métricas específicas, basada en el proceso de evaluación propuesto por la norma **ISO/IEC 25040** y las características de calidad del modelo **ISO/IEC 25010**, particularmente la subcaracterística de *tolerancia a fallos* dentro del atributo de *fiabilidad*. Estas métricas permiten analizar tanto

el diseño técnico (por ejemplo, uso de bloques try/catch y patrones de captura de errores), como la experiencia del usuario ante fallos controlados. La tabla contempla criterios cuantificables y escalas de valoración que clasifican el cumplimiento en cuatro niveles: *inaceptable, aceptable, rango objetivo* y *excede expectativas*. Esta estratificación permite no solo medir el grado actual de conformidad con los estándares esperados, sino también identificar áreas de mejora concreta. Por ejemplo, un sistema que registre el 90% de sus errores con contexto estructurado, que utilice bloques de manejo de errores en la mayoría de sus componentes críticos, y que nunca exponga mensajes técnicos crudos al usuario, puede considerarse que excede los estándares de calidad esperados en esta dimensión. En conjunto, esta metodología constituye una herramienta práctica, replicable y alineada con marcos normativos internacionales, permitiendo verificar con rigor si el producto cumple con estándares de calidad robustos en lo que respecta al tratamiento de errores.

| Criterio Evaluado | Métrica | Inaceptable | Aceptable | Rango Objetivo | Excede Expectativas |
|---|---|---|---|---|---|
| Captura de errores en bloques críticos | % de funciones críticas con try/catch | < 40% | 40–69% | 70–89% | ≥ 90% |
| Registro estructurado en logs | % de excepciones logueadas con contexto | < 50% | 50–69% | 70–85% | ≥ 86% |
| Estabilidad del sistema ante errores | N° de caídas por errores no controlados | > 5 en pruebas | 3–5 en pruebas | 1–2 en pruebas | 0 |
| Mensajes de error amigables al usuario | % de errores que devuelven mensaje claro y útil | < 50% | 50–74% | 75–89% | ≥ 90% |

# Pruebas Automatizadas

## Descripción

Deben existir pruebas unitarias, de integración y/o funcionales automatizadas.

## Forma que vamos a usar para medir y valores aceptados

En el contexto del aseguramiento de la calidad del software, las **pruebas automatizadas** representan un componente central para garantizar que el sistema funcione correctamente

frente a cambios, refactorizaciones y nuevas funcionalidades. Su implementación permite detectar errores de manera temprana, reducir costos de mantenimiento y mantener un ciclo de desarrollo ágil y sostenible. El producto debe incorporar **pruebas unitarias, de integración y/o funcionales automatizadas**, con el objetivo de validar tanto el comportamiento de componentes individuales como la interacción entre ellos y el cumplimiento de los requerimientos funcionales completos. Esta estrategia se encuentra alineada con las buenas prácticas promovidas por la norma **ISO/IEC 25010**, particularmente dentro de las características de *adecuación funcional* y *mantenibilidad*, y se operacionaliza mediante el enfoque de evaluación sistemática definido por **ISO/IEC 25040**.

Para ello, se propone una metodología clara y práctica basada en métricas cuantificables que permiten analizar el grado de cobertura de código, el nivel de integración de las pruebas en el proceso de desarrollo, la detección de regresiones y la existencia de reportes automatizados que documenten los resultados de las pruebas. A continuación, se presenta una tabla que resume los criterios establecidos para la evaluación de esta dimensión de calidad, clasificados en distintos niveles de cumplimiento: *inaceptable*, *aceptable*, *rango objetivo* y *excede expectativas*. Esta clasificación busca facilitar el diagnóstico del estado actual del producto en términos de automatización del testing y trazar un camino de mejora continua hacia estándares de calidad más altos.

| Criterio Evaluado | Métrica | Inaceptable | Aceptable | Rango Objetivo | Excede Expectativas |
|---|---|---|---|---|---|
| **Cobertura de código (unitarias)** | % de líneas de código cubiertas por pruebas unitarias | < 30% | 30–59% | 60–85% | > 85% |
| **Cobertura de flujos críticos (funcionales)** | % de funcionalidades principales cubiertas por pruebas E2E | < 50% | 50–74% | 75–89% | ≥ 90% |
| **Integración en CI/CD** | ¿Se ejecutan automáticamente en cada commit o merge? | No | Parcial (manual) | Sí, con fallos ocasionales | Sí, estable y con reportes |

| | | | | | |
|---|---|---|---|---|---|
| **Detección de regresiones** | N° de regresiones no detectadas antes del despliegue | > 5 | 3–5 | 1–2 | 0 |
| **Reportes y trazabilidad de resultados** | ¿Se generan reportes automatizados con resultados de pruebas? | No | Sí, pero incompletos | Sí, con métricas básicas | Sí, detallados y vinculados a tickets/issues |

# Desacoplamiento entre Componentes

## Descripción

Se deben utilizar patrones de diseño que minimicen dependencias directas (como inyección de dependencias).

## Forma que vamos a usar para medir y valores aceptados

Para medir el desacoplamiento entre los componentes, hay que tener en cuenta los siguientes criterios para cada componente:

- **Fan-in:** Cuántos componentes dependen de este componente.
- **Fan-out:** Cuántos componentes este componente depende.

La escala de cada categoría es:

- **Fan-in:** $Componentes\ dependientes\ del\ componente\ /\ Componentes\ totales.$
- **Fan-out:** $Componentes\ que\ el\ componente\ depende\ /\ Componentes\ totales.$

El nivel de desacoplamiento de un componente es $1\ -\ (Fan\ in\ +\ Fan\ out)\ /\ 2.$
El nivel de desacoplamiento del producto es
$(\sum Nivel\ de\ desacoplamiento\ del\ componente)\ /\ Componentes\ totales.$
A partir del resultado final, establecemos los valores para la escala de la métrica según el tamaño el proyecto a analizar:

1. **Proyecto Chico:** La escala de un proyecto de menor tamaño no requiere de un gran desacoplamiento de componentes.
    a. Satisfactorio
        i. Excede los requerimientos: Mayor o igual a 0.85
        ii. Rango aceptado: Mayor o igual a 0.75
        iii. Mínimamente Aceptable: Mayor o igual a 0.65
    b. Insatisfactorio
        i. Inaceptable: Menor a 0.65

2. **Proyecto Mediano:** Un proyecto con un tamaño mediano implica llevar un mayor desacople para tener mayor independencia .
    a. Satisfactorio
        i. Excede los requerimientos: Mayor o igual a 0,90
        ii. Rango aceptado: Mayor o igual a 0,85
        iii. Mínimamente Aceptable: Mayor o igual a 0.75
    b. Insatisfactorio
        i. Inaceptable: Menor a 0.75
3. **Proyecto Grande:** Un proyecto grande requiere de mucho desacople de componentes para evitar la mayor cantidad de dependencias posibles.
    a. Satisfactorio
        i. Excede los requerimientos: 1.0
        ii. Rango aceptado: Mayor o igual a 0,90
        iii. Mínimamente Aceptable: Mayor o igual a 0,85
    b. Insatisfactorio
        i. Inaceptable: Menor a 0,85

# Estandarización del Código

## Descripción

El código debe seguir convenciones claras y estándares definidos (nombres, estructura, estilo).

## Forma que vamos a usar para medir y valores aceptados

**Analizando las líneas de código entregables del sistema podemos analizar los lenguajes de programación utilizados en el proyecto y de esta forma poder asociar las guías de estilo que normalmente tiene en cuenta la comunidad de programadores de esos lenguajes:**

- Python: PEP8.
- JavaScript: Airbnb JavaScript Style Guide.
- Java: Google Java Style Guide.
- C#: Microsoft C# Coding Conventions.
- C++: Google C++ Style Guide o LLVM Coding Standards.
- etc.

**Dependiendo el lenguaje podemos analizar las líneas de código entregables usando el linter (analizador estático) correspondiente a cada lenguaje, buscando que nos de la cantidad de líneas de código entregables que presenten violaciones de los estándares normalmente utilizados.**

**La forma de medir la estandarización de código será:**
$1 - (Líneas\ con\ violaciones\ /\ total\ de\ líneas\ de\ código\ entregables)$. A partir del resultado de esa fórmula, establecemos los valores para la escala de la métrica según el tamaño el proyecto a analizar:

1. **Proyecto Chico:** Un proyecto pequeño que no requiere un mayor desarrollo y normalmente con uno o 2 lenguajes de programación debe preservar una buena estandarización
    a. Satisfactorio
        i. Excede los requerimientos: 1.0
        ii. Rango aceptado: Mayor o igual a 0,90
        iii. Mínimamente Aceptable: Mayor o igual a 0,80
    b. Insatisfactorio
        i. Inaceptable: Menor a 0,80
2. **Proyecto Mediano:** Un proyecto con un número mayor de funcionalidades y desarrollo con varios lenguajes de programación y archivos puede tener errores menores no críticos en su estandarización y estilo de código
    a. Satisfactorio
        i. Excede los requerimientos: Mayor o igual a 0,90
        ii. Rango aceptado: Mayor o igual a 0,80
        iii. Mínimamente Aceptable: Mayor o igual a 0.70
    b. Insatisfactorio
        i. Inaceptable: Menor a 0.70
3. **Proyecto Grande:** La escala de un proyecto de mayor tamaño hace que aparezcan grandes porciones de código que, a veces, son heredadas o externas (algo que no tuvimos control sobre sus estilos), en estos casos vamos a tolerar errores menores de estandarización y estilos en el código.
    a. Satisfactorio
        i. Excede los requerimientos: Mayor o igual a 0.85
        ii. Rango aceptado: Mayor o igual a 0.75
        iii. Mínimamente Aceptable: Mayor o igual a 0.60
    b. Insatisfactorio
        i. Inaceptable: Menor a 0.60

# Control de Versiones y Trazabilidad de Cambios

## Descripción

Todo cambio al código debe registrarse adecuadamente en el sistema de control de versiones.

## Idea general

Los mensajes de commit serán claros, describirán el cambio realizado y estarán vinculados a diferentes tareas o issues. Esto permitirá mantener la trazabilidad de los cambios a lo largo del proyecto y facilitará el trabajo en equipo.

**Forma que vamos a usar para medir y valores aceptados:**

- **Analizamos cada commit siguiendo estos criterios:**
    - Claridad: El commit explica claramente que se hizo.

    - Referencia a Tarea/issue: El commit menciona una tarea o issue que se relacione al mismo.
    - Frecuencia correspondiente: Los commits se hacen con una frecuencia adecuada para el proyecto de modo que no sean muy dispersos.
    - Tamaño correcto: Los commits son de un tamaño adecuado y no son demasiado grandes.
- **A cada commit le asignamos un puntaje:**
    - Aceptable (1.0 puntos): Es claro, con referencia a una tarea, tamaño adecuado y con una frecuencia adecuada.
    - Intermedio (0.5 puntos): Mensaje claro pero sin referencia a tarea o con tamaño ligeramente inadecuado.
    - Inadecuado (0.25 puntos): Mensaje poco claro, sin referencia, demasiado grande y con frecuencia muy dispersa.
- **Finalmente aplicamos una fórmula para obtener el promedio de calidad de versiones:**
    - $Puntaje\ Total\ =\ (Suma\ de\ puntos\ de\ los\ commits)\ /\ (Cantidad\ total\ de\ los\ commits)$

**Métrica según el tamaño del proyecto:**

- **Proyecto Chico:**
    - Excede los requerimientos: ≥ 0.95
    - Rango aceptado: ≥ 0.80
    - Mínimamente aceptable: ≥ 0.65
    - Inaceptable: < 0.65
- **Proyecto Mediano:**
    - Excede los requerimientos: ≥ 0.90
    - Rango aceptado: ≥ 0.75
    - Mínimamente aceptable: ≥ 0.60
    - Inaceptable: < 0.60
- **Proyecto Grande:**
    - Excede los requerimientos: ≥ 0.85
    - Rango aceptado: ≥ 0.70
    - Mínimamente aceptable: ≥ 0.55
    - Inaceptable: < 0.55