CARRERA DE COMPUTACIÓN

1. Datos Informativos

1.1. Módulo: 1

1.2. Nivel: 7

1.3. Apellidos y Nombres: Solis Ramírez Aida Milena – Valenzuela Quisiquinga Cristian Sebastian – Valdivieso Aslalema Jeremy Joel

1.4. Tema: Métricas de Gestión de proyectos de software

1.5. Fecha: 19-nov-2025

2. Objetivo

Analizar las métricas fundamentales en la gestión de proyectos de software para comprender su impacto en la estimación de costos, el control de calidad y la toma de decisiones estratégicas durante el ciclo de vida del desarrollo.

3. Contenido

Introducción

Para el desarrollo de este informe, iniciamos la investigación partiendo de la premisa de que la medición es un elemento insustituible en cualquier ingeniería. Comprendiendo que, sin métricas, la gestión es meramente subjetiva. Tal como lo indicado en clase las métricas de software se refieren a una amplia gama de medidas para el software de computadora.

Durante la revisión bibliográfica, se encontró que Roger S. Pressman, en su libro *Ingeniería del software: un enfoque práctico*, establece una distinción clara que guía el análisis:

"Las métricas del proyecto se utilizan para minimizar la planificación del desarrollo haciendo los ajustes necesarios que eviten retrasos y reduzcan problemas y riesgos potenciales. Las métricas del producto se utilizan para evaluar la calidad de los entregables" (Pressman, 2010).

Basándonos en esta definición, enfocamos mi trabajo en cómo estos datos cuantitativos permiten a los gestores evaluar la eficacia de las técnicas y herramientas utilizadas. Asimismo, consultando la obra de Ian Sommerville, quien argumenta que la gestión de proyectos de software es esencial porque la ingeniería de software siempre está sujeta a restricciones de presupuesto y calendario (Sommerville, 2011). Por lo tanto, se estableció que el uso de métricas no es opcional, sino un requisito para asegurar que el software cumpla con las expectativas del cliente dentro de los límites establecidos.

Desarrollo

Definición de Medición y Métricas

Para asentar las bases teóricas de este trabajo, primero se estableció la distinción conceptual entre "medición" y "métrica", ya que a menudo se confunden. Al consultar la obra *Software Metrics: A Rigorous and Practical Approach*, se definió la medición como el proceso de asignar números o símbolos a los atributos de entidades en el mundo real de tal manera que se describan de acuerdo con reglas claramente definidas (Fenton & Pfleeger, 1997). Entendiendo que la medición es el dato bruto, el valor cuantitativo directo, por ejemplo, el número de líneas de código.

Posteriormente, se analizó el concepto de métrica. Determinando que una métrica es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. Según lo explican los expertos, la métrica relaciona las medidas individuales para proporcionar una visión más profunda; es decir, transformé los datos en información útil para la toma de decisiones.

¿Por qué Medir el software?

Durante la investigación, se indagó sobre la necesidad crítica de aplicar estas mediciones. Encontrando que la razón fundamental radica en la gestión y la mejora. Tal como lo cita Roger Pressman, existe una máxima en la ingeniería: "No se puede controlar lo que no se puede medir" (Pressman, 2010).

Se identificó que medimos el software para:

1. **Caracterizar:** Comprendiendo el estado actual de los procesos y productos.
2. **Evaluar:** Se determinó si se cumplieron los estándares de calidad.
3. **Predecir:** Utilizando datos históricos para estimar esfuerzos futuros.
4. **Mejorar:** Se localiza ineficiencias para optimizar el desarrollo.

Sin estas mediciones, se concluye que la gestión de proyectos se basaría únicamente en evaluaciones subjetivas y no en datos objetivos.

Categorías Básicas de las Métricas

Basándose en la clasificación estándar de la ingeniería de software, organizamos las métricas en tres dominios fundamentales que interactúan entre sí:

- **Métricas de Proceso:** Centrándonos en el comportamiento del desarrollo en sí mismo. Se recopiló datos sobre el tiempo de ciclo y la eficacia en la eliminación de defectos. El objetivo que perseguimos al estudiar estas métricas fue la mejora a largo plazo del flujo de trabajo y la madurez de la organización de software.
- **Métricas de Proyecto:** Se las utilizó para describir el estado del proyecto en curso. Evaluando variables tácticas como el costo actual, el cronograma, el esfuerzo y el personal asignado. Determinamos que estas métricas son esenciales para que el gestor del proyecto realice ajustes inmediatos y evite desviaciones en la planificación.
- **Métricas de Producto:** Finalmente, se examinó las características del software entregable. Enfocándonos en medir atributos tangibles como el tamaño (Líneas de Código o Puntos de Función), la complejidad ciclomática, el rendimiento y la calidad que vendría siendo la tasa de fallos. Considerando que estas métricas son vitales para asegurar que el producto final satisfaga los requisitos del cliente.

Caso de estudio

Caso de Estudio

- Caso simulado: “Un equipo ha desarrollado una app educativa. Deben evaluar cómo va el proyecto. Les entrego los siguientes datos”
- Datos
  - LOC: 4,800
  - Número de módulos: 12
  - Tiempo promedio para corregir errores: 30 horas
  - Costo planeado: $5,000 / Costo real: $5,600
  - Bugs reportados: 10 en fase de pruebas sí

Para aplicar los conceptos teóricos, analizamos un caso simulado de un desarrollo de software educativo. Examinando un conjunto de datos brutos proporcionados por el equipo de desarrollo, que incluía 4,800 líneas de código (LOC), 12 módulos, un tiempo promedio de corrección de errores de 30 horas, y una discrepancia entre el costo planeado ($5,000) y el real ($5,600).

A partir de esta información, identificamos y clasificamos las siguientes métricas según su categoría:

a) **Métrica de Producto: Densidad de Defectos** Se utilizó los datos de "LOC: 4,800" y "Bugs reportados: 10" para calcular la calidad del código entregado. Siguiendo la fórmula estándar citada por Pressman, calculamos la densidad de defectos dividiendo el número de errores conocidos por el tamaño del software en miles de líneas de código (KLOC).

***Cálculo realizado**:* 10 errores / 4.8 KLOC = **2.08 errores por KLOC**. Interpretamos este valor como un indicador directo de la calidad del producto en su fase de pruebas.

b) **Métrica de Proceso: Tiempo Medio de Reparación (MTTR)** Para evaluar el proceso de mantenimiento, se seleccionó el dato de "Tiempo promedio para corregir errores: 30 horas". Al contrastar esto con la literatura, observamos que esta métrica refleja la

eficiencia del equipo en la fase de depuración. Tal como indica Sommerville, un tiempo de reparación elevado suele ser síntoma de un código difícil de entender (baja mantenibilidad) o de un proceso de diagnóstico ineficiente (Sommerville, 2011).

c) **Métrica de Proyecto: Variación del Costo (CV)** Finalmente, se evaluó la salud financiera del proyecto comparando el "Costo planeado ($5,000)" con el "Costo real ($5,600)".

Análisis: Identificamos una desviación negativa de $600, lo que representó un sobrecosto del 12%. Esta métrica de proyecto me permitió concluir que hubo fallos en la estimación inicial o imprevistos no mitigados durante la ejecución.

Propuesta de Mejora Basada en los Datos

Tras analizar los indicadores, se detectó que el punto más crítico no fue el sobrecosto en sí mismo, sino el tiempo promedio de corrección de 30 horas, el cual consideramos excesivo para un proyecto de tamaño pequeño (4.8 KLOC).

Propusimos la implementación de Revisiones Técnicas Formales (RTF).

Basando esta propuesta en la evidencia presentada por Roger Pressman, quien afirma que las revisiones técnicas son el filtro de calidad más efectivo en la ingeniería de software (Pressman, 2010). Se dedujo que las 30 horas invertidas en corregir cada error sugerían que los desarrolladores tardaban demasiado en localizar la causa raíz del fallo, probablemente debido a un código complejo o mal documentado.

Para lo cual se argumenta que, al aplicar revisiones de pares antes de la fase de pruebas:

1. Se reduciría la inyección de errores latentes.
2. Se mejoraría la legibilidad del código, disminuyendo drásticamente esas 30 horas de corrección en el futuro.
3. Como consecuencia directa, se controlaría la desviación del costo, ya que el retrabajo es la causa principal de los sobrecostos en proyectos de software.

4. Conclusiones

Al finalizar este trabajo de investigación y análisis práctico sobre las métricas de gestión de proyectos de software, llegamos a las siguientes conclusiones:

Diferenciación Clave: Comprendimos que existe una diferencia sustancial entre "medición" y "métrica". Mientras que la medición nos proporcionó datos brutos (como el número de líneas de código o costo real), la métrica fue la interpretación que dio sentido a esos datos. Concluimos, basándonos en Fenton y Pfleeger, que, sin esta transformación de dato a información, la toma de decisiones gerenciales carecería de fundamento lógico.

Interdependencia de Categorías: Se observó que las métricas de proceso, proyecto y producto no funcionan de forma aislada. En el caso de estudio, pudimos constatar cómo una métrica de proceso deficiente (el tiempo excesivo de corrección de errores) impactó directamente y de forma negativa en una métrica de proyecto (generando sobrecostos). Esto nos permitió afirmar que descuidar la calidad del proceso de ingeniería inevitablemente degrada la salud financiera del proyecto.

El Valor de la Prevención: A través de la propuesta de mejora, se ratifica la teoría de Roger Pressman sobre el costo de la calidad. Determinando que invertir esfuerzo en métricas preventivas, como las revisiones técnicas formales, es infinitamente más rentable que gestionar métricas correctivas al final del ciclo de vida. El análisis de los datos demostró que "arreglar" el software es costoso y lento si no se mide la calidad desde el inicio.

Control y Visibilidad: Finalmente, consolidamos la idea de que la ingeniería de software requiere disciplina cuantitativa. Verificando que las métricas actúan como el tablero de control del gestor; sin ellas, el estado del proyecto es invisible y, por tanto, ingobernable.

Fenton, N., & Pfleeger, S. (1997). *Software Metrics: A Rigorous and Practical Approach* (Vol. 2). Boston: PWS Publishing.

Pressman, R. (2010). *Ingeniería del software: un enfoque práctico* (Vol. 7ma). Columbus, Ohio: McGraw-Hill.

Sommerville, I. (2011). *Ingeniería del Software* (Vol. 9). México, México: Pearson Educación.