Métricas del Proceso

1. **Tiempo total del proceso (Lead Time)**
   - **Definición:** Tiempo total desde que se solicita una tarea hasta que se entrega.
   - **Fórmula:** Tiempo total de proceso = Fecha de Entrega − Fecha de Solicitud
   - **Ejemplo:** Si un requerimiento fue solicitado el 1 de septiembre y se entregó el 10 de septiembre, el Plazo de entrega es de 9 días.

     Plazo de entrega = 10 − 1

2. **Tiempo de Ciclo**
   - **Definición:** Tiempo desde que se comienza a trabajar en una tarea hasta su finalización.
   - **Fórmula:** Tiempo de Ciclo = Fecha de Finalización – Fecha de Solicitud.
   - **Ejemplo:** Si se comienza a trabajar en una tarea el 5 de septiembre y se termina el 8 de septiembre, el tiempo de ciclo es de 3 días.

     Tiempo de Ciclo = 8 – **5**

¿Plazo de entrega es igual a Tiempo de Ciclo?

No, **Plazo de entrega** y **Tiempo de Ciclo** son conceptos relacionados, pero no son lo mismo.

- **Plazo de entrega:** Es el tiempo total desde que se solicita un trabajo hasta que se entrega, incluyendo tiempos de espera.
- **Tiempo de Ciclo:** Es el tiempo que se tarda en trabajar en una tarea desde que se inicia hasta que se completa, excluyendo tiempos de espera.

Ejemplo: Si un requerimiento se solicita el 1 de septiembre y se entrega el 10 de septiembre (incluyendo esperas), el **Plazo de entrega** es de 9 días. Si el trabajo comienza el 5 de septiembre y termina el 8 de septiembre, el **Tiempo de Ciclo** es de 3 días.

¿Qué es el Tiempo de Espera?

El **Tiempo de Espera** se refiere al período en el que una tarea o actividad está en cola o pendiente sin que nadie esté trabajando activamente en ella. Este tiempo no incluye el tiempo de trabajo activo, solo el tiempo en que la tarea está "esperando" a ser abordada.

Ejemplo:

- Si una tarea se solicita el 1 de septiembre, se comienza a trabajar en ella el 5 de septiembre y se finaliza el 10 de septiembre:
  - **Plazo de entrega:** 9 días (del 1 al 10 de septiembre). Plazo de entrega = 10 − 1
  - **Tiempo de Ciclo:** 5 días (del 5 al 10 de septiembre). Tiempo de Ciclo = 10 – 5
  - **Tiempo de Espera:** 4 días (del 1 al 5 de septiembre). Tiempo de espera = 5 - 1

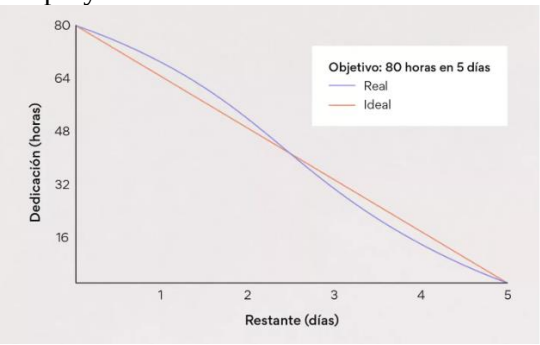
3. **Número de Rework (Re-trabajo)**
   - **Definición:** Tiempo dedicado a corregir errores después de la implementación inicial.
   - **Fórmula:** Tiempo total adicional dedicado a correcciones.
   - **Ejemplo:** Si se tarda 5 horas en corregir errores en un módulo de software, el rework es de 5 horas.

4. **Eficiencia del Proceso**
   - **Definición:** Mide qué tan eficientemente se está utilizando el tiempo en actividades productivas.
   - **Fórmula:** Eficiencia = (Tiempo Productivo / Tiempo Total) ×100
   - **Ejemplo:** Si de 8 horas de trabajo, 6 horas son productivas, la eficiencia es del 75%.

     Eficiencia = (6 / 8) ×100

5. **Gráfico de Burndown**
   - **Definición:** Visualización que muestra la cantidad de trabajo restante contra el tiempo.
   - **Ejemplo:** Un gráfico de Burndown que muestra que el equipo está completando tareas más lentamente de lo planeado indica que podrían no cumplir con la fecha de entrega.

Burndown chart es el término en inglés para referirse a un gráfico burndown o gráfico de trabajo pendiente en el entorno de Scrum. Se trata de un diagrama que muestra el trabajo que queda por hacer con respecto al tiempo necesario para completarlo. Puede ser de gran utilidad para los equipos que trabajan en sprints, ya que es una forma efectiva de visualizar si se puede cumplir con los plazos a medida que avanza el proyecto.



Métricas del Proyecto

1. **Cumplimiento de Cronograma o porcentaje de tareas completadas (índice de entrega a tiempo)**
   - **Definición:** Mide el porcentaje de tareas completadas a tiempo respecto al total planificado.
   - **Fórmula:** Cumplimiento = (Tareas a Tiempo / Tareas Planificadas) ×100
   - **Ejemplo:** Si de 10 tareas se completaron 8 a tiempo, el cumplimiento es del 80%.
     Cumplimiento = (8 / 10) ×100

2. **Control de Costos**
   - **Definición:** Evalúa cómo se está manejando el presupuesto del proyecto en relación con los costos reales.
   - **Fórmula:** Monitoreo continuo del presupuesto vs. los gastos reales.
   - **Ejemplo:** Si el presupuesto es de $10,000 y se han gastado $8,000, se ha gastado el 80% del presupuesto. 8000/100

3. **Desviación de Costos**
   - **Definición:** Mide la diferencia entre los costos estimados y los reales.
   - **Fórmula:** Desviación = ((Costo (presupuesto) Real – Costo (presupuesto) Estimado) / Costo (presupuesto) Estimado) * 100

Valor ganado – costo real

   - **Ejemplo:** Si el costo estimado era de $5,000 y el real fue $6,000, la desviación es del 20%.
     Desviación = ((6000 − 5000) / 5000) * 100

4. **Horas Hombre**
   - **Definición:** Tiempo de trabajo invertido por una persona para completar una tarea.

- **Fórmula:** Horas Hombre = total de personas * horas trabajadas.
- **Ejemplo:** Un proyecto que requiere 5 personas trabajando 20 horas cada una.
  Horas hombre = 5 * 20

5. **Desviación de Tiempo**

   Formula: Desviación de Tiempo = Tiempo Real − Tiempo Estimado

Métricas de Producto

1. Densidad de Defectos verificar si pertenece a proyecto

- **Definición:** Número de defectos por línea de código o módulo.
- **Fórmula:** Densidad = (Numero de Defectos / Líneas de Código)
- **Ejemplo:** Si se encuentran 5 defectos en 1,000 líneas de código, la densidad es de 0.005 defectos por línea.

  Densidad = 5 / 1000

2. **Cobertura de Pruebas:**
   Cobertura de Pruebas = (Líneas Cubiertas por Pruebas / Líneas de Código) × 100

3. **Productividad del Código:**
   Productividad = (Líneas de Código / Horas de Desarrollo)

4. **Mantenibilidad**
   - **Definición:** Medida de qué tan fácil es modificar y corregir el código.
   - **Fórmula:** A menudo evaluada mediante herramientas que analizan la complejidad y claridad del código.
   - **Ejemplo:** Código con menor complejidad ciclomática y buena documentación es más mantenible.

Herramientas para medir la Mantenibilidad
Existen herramientas para medir la mantenibilidad gratuitas como de pago.
SonarQube:
- **Versión Gratuita:** Disponible con funcionalidades básicas.
- **Cómo se utiliza:** Se configura un servidor SonarQube y se integran los proyectos mediante scripts de análisis en lenguajes como Java, JavaScript, etc. Proporciona un dashboard con métricas de calidad, detectando duplicaciones de código, complejidad ciclomática, violaciones de estilo, etc.

Checkstyle, PMD (Java):
- **Gratuitas:** Plugins de análisis estático, principalmente para Java.
- **Cómo se utilizan:** Se ejecutan desde el IDE o como parte del pipeline de CI/CD, revisando la estructura y reglas del código.

Code Climate:
- **Versión Gratuita:** Limitada a proyectos open-source.
- **Cómo se utiliza:** Se integra con repositorios como GitHub y analiza el código en cada push o pull request, generando un reporte de mantenibilidad.

5. **Índice de Complejidad del Código (Ciclomática)**

- **Definición:** Evalúa la complejidad del flujo de control del código.
- **Fórmula:**

  Complejidad = E – N + 2P

dónde:

E= Número de aristas (líneas de control de flujo)**.** Las aristas son las conexiones entre los nodos que muestran el flujo del programa.

N = Número de nodos (bloques de código)

P = Número de componentes conectados (generalmente 1 para un solo programa)

Ejemplo: La **Complejidad Ciclomática** de un código se calcula mediante el grafo de control de flujo:
Ejemplo: Considera el siguiente código:

```
int calculate(int a, int b) {
int result = 0;
if (a > b) {
        result = a - b; }
else if (a == b) {
        result = 0; }
else {
        result = b - a; }
return result; }
```

Nodos (N): 5 (inicio, if, else-if, else, return).

Aristas (E): 6 (inicio -> if, if -> else-if, else-if -> else, etc.).

Componentes (P): 1 (función única).

Complejidad Ciclomática = $6 – 5 + 2 \times 1 = 3$

Un valor de 3 sugiere que hay 3 posibles caminos de ejecución, lo cual da una idea de cuántas pruebas se necesitan para cubrir todos los casos.

6. Acoplamiento y Cohesión (dependencia de funciones o complejidad estructural)

- **Definición:** Mide la interdependencia entre módulos (acoplamiento) y la unión de funciones dentro de un módulo (cohesión).
- **Fórmula:** Evaluada mediante análisis de dependencias y relaciones.
- **Ejemplo:** Módulos con bajo acoplamiento y alta cohesión son ideales.

Cohesión: Mide cuán estrechamente relacionadas están las funcionalidades dentro de un módulo.
LCOM = Número de Métodos − Número de Métodos que utilizan atributos comunes

Ejemplo: Un módulo con dos métodos, donde uno usa las variables ´x´ y ´y´ y el otro solo ´y´, tendrá baja cohesión.

Acoplamiento: Mide la interdependencia entre módulos. Puede ser contado como el número de conexiones (referencias) entre módulos.

Ejemplo: Si un módulo A llama funciones de un módulo B varias veces, hay un acoplamiento alto entre A y B.

¿Qué significa Lack of Cohesion of Methods (LCOM)?
Lack of Cohesion of Methods (LCOM) mide la falta de cohesión dentro de una clase en programación orientada a objetos. Un LCOM alto indica que los métodos de la clase no están utilizando los mismos atributos, lo cual sugiere que la clase realiza más de una función y podría necesitar ser dividida.

Ejemplo:
- Clase con 3 métodos:
  Método 1 usa atributos x y y.
  Método 2 usa y y z.
  Método 3 usa solo x.

- **LCOM:** 3 métodos - 1 grupo de métodos con atributos comunes = 2.

Un LCOM alto indica baja cohesión, sugiriendo la refactorización de la clase.
Ejemplo utilizando código

```
int calculate(int a, int b) {
   int result = 0;
   if (a > b) {
      result = a - b;
   } else if (a == b) {
      result = 0;
   } else {
      result = b - a;
   }
   return result;
}
```

- **Nodos (N)**: 5 (inicio, if, else-if, else, return).
- **Aristas (E)**: 6 (inicio --> if, if --> else-if, else-if --> else, etc.).
- **Componentes (P)**: 1 (función única).

Complejidad Ciclomatica = $6 – 5 + 2 \times 1 = 3$

Un valor de 3 sugiere que hay 3 posibles caminos de ejecución, lo cual da una idea de cuántas pruebas se necesitan para cubrir todos los casos.
¿En la Métrica de Halstead, int y = son Operadores?
Sí, en la Métrica de Halstead:
- **Operadores:** Son símbolos y palabras reservadas que representan operaciones, como int, =, +, if, return.

- **Operandos:** Son las entidades sobre las cuales actúan los operadores, como variables (a, b), constantes (5), y valores de retorno.

En el ejemplo int sum = a + b
- **Operadores (n1n_1n1)**: {int, =, +} = 3 operadores.
- **Operandos (n2n_2n2)**: {sum, a, b} = 3 operandos.

7. **Métrica de Halstead**
   - **Definición:** Evalúa la complejidad del software mediante operadores y operandos.
   - **Fórmulas:**
     Longitud: n1 + n2
     Volumen: (n1+n2) log2(n1+n2)
     Dificultad: (n1/2) ×N2/n2
   - **Ejemplo:** Software con menor volumen y dificultad es menos propenso a errores.

La Métrica de Halstead utiliza operadores y operandos para medir la complejidad del software:
- **n1 (Operadores Únicos):** Número de operadores distintos utilizados en el código (ej., +, -, if, for).
- **n2 (Operandos Únicos):** Número de operandos distintos utilizados (ej., variables, constantes).

Ejemplo: Para el siguiente código:

int sum = a + b;
- Operadores únicos (n1): {int, =, +} = 3
- Operandos únicos (n2): {sum, a, b} = 3

Métricas Relacionadas con los Recursos

1. **Costo por Hora de Desarrollo**
   - **Definición:** Mide el costo del desarrollo dividiendo el costo total entre las horas de desarrollo.
   - **Fórmula:** Costo por Hora = Costo Total / Horas de Desarrollo
   - **Ejemplo:** Si el costo total es de $20,000 y se trabajaron 1,000 horas, el costo es $20 por hora.

2. **Productividad del Desarrollador**
   - **Definición:** Mide la cantidad de funcionalidades entregadas por desarrollador en un periodo. Mide la cantidad de entregables producidos en relación con los recursos utilizados.
   - **Fórmula:** Productividad = Funcionalidades Entregadas / Horas Hombre Invertidas
   - **Ejemplo:** Un equipo entrega 10 funcionalidades y se invirtieron 50 horas hombre:
   - Productividad = 10 / 50 = 0.2 funcionalidades por hora

3. **Velocidad del Equipo:**
   Velocidad = Funcionalidades Entregadas / (Número de Desarrolladores × Tiempo (meses))

4. **Uso de Recursos Computacionales**
   - **Definición:** Evalúa el uso de CPU, memoria y otros recursos.

- **Ejemplo:** Si una aplicación usa constantemente el 80% de la CPU, podría necesitar optimización.

Métricas Adicionales

1. **Índice de Satisfacción del Cliente**
   - **Definición:** Mide la percepción del cliente sobre la calidad del producto.
   - **Fórmula:** Evaluado mediante encuestas de satisfacción.
   - **Ejemplo:** Un índice de satisfacción del 90% indica alta aceptación del producto.

2. **Tasa de Entregas a Tiempo**
   - **Definición:** Mide la proporción de entregas que cumplen los plazos.
   - **Fórmula:** Tasa de Entregas = (Entregas a Tiempo / Total de Entregas) ×100
   - **Ejemplo:** Si 18 de 20 entregas se realizaron a tiempo, la tasa es del 90%.

3. **Índice de Documentación Completa**
   - **Definición:** Mide el nivel de documentación en comparación con lo requerido.
   - **Fórmula:** Índice = (Documentos Completos / Documentos Necesarios) ×100
   - **Ejemplo:** Si se completaron 7 de 10 documentos requeridos, el índice es del 70%.

¿Qué se utiliza para medir la Velocidad del Equipo?
La **Velocidad del Equipo** se mide generalmente utilizando **Puntos de Historia** o **Tareas Completadas** durante un Sprint o Iteración.

- **Puntos de Historia:** Miden la complejidad y esfuerzo de las tareas, asignados a cada tarea o funcionalidad por el equipo.
- **Tareas Completadas:** El número total de tareas que el equipo ha terminado durante el Sprint.

Ejemplo: Si un equipo completa 30 puntos de historia en una iteración de dos semanas, su velocidad es de 30 puntos.

¿Qué herramientas se utilizan para medir la Mantenibilidad?
La mantenibilidad se evalúa usando herramientas de análisis estático de código que calculan métricas como la complejidad, la claridad, y la modularidad del código. Algunas herramientas comunes son:

- **SonarQube:** Evalúa la mantenibilidad mediante la detección de código duplicado, complejidad ciclomática, y código mal estructurado.
- **Code Climate:** Analiza la calidad del código y proporciona un puntaje de mantenibilidad.
- **Checkstyle y PMD:** Detectan violaciones de estilo y problemas estructurales que afectan la mantenibilidad.