**GUÍA DE APRENDIZAJE 2 CLASIFICACIÓN DE METRICAS**

**JENNIFER ANDREA ROMANI JAMAICA**

**JUAN DAVID SABOYÁ JIMENEZ**

**SERVICIO NACIONAL DE APRENDIZAJE (SENA)**

**CENTRO DE DISEÑO Y METROLOGÍA**

**ANÁLISIS Y DESARROLLO DE SISTEMAS DE INFORMACIÓN**

**FICHA 683019**

**BOGOTÁ D.C.**

**2015**

**Tabla de contenido**

[Realizar la medición a 6 software conocidos, utilizando las imágenes del programa paso a paso. 3](#_Toc427061166)

[Investigar métricas de la calidad de software, 2 ejemplos 13](#_Toc427061167)

[Investigar métricas para el esquema de puntuación, 2 ejemplos 14](#_Toc427061168)

[Investigar Factores de calidad ISO 9126 14](#_Toc427061169)

[Investigar estructura para las métricas de software 19](#_Toc427061170)

[Investigar métricas del modelo de análisis 20](#_Toc427061171)

[Investigar métricas Bang (de marco) 22](#_Toc427061172)

[Investigar métricas de la calidad de especificación 28](#_Toc427061173)

[Investigar métricas del modelo de diseño 30](#_Toc427061174)

[Investigar métricas de diseño de alto nivel 31](#_Toc427061175)

[Investigar métricas del código fuente 32](#_Toc427061176)

[Investigar métricas para la prueba 34](#_Toc427061177)

[Investigar métricas del mantenimiento 37](#_Toc427061178)

[Investigar métricas para predecir la calidad 38](#_Toc427061179)

[Investigar métricas del producto 39](#_Toc427061180)

[Investigar métricas de medición del proceso 43](#_Toc427061181)

[Investigar parámetros involucrados en el costo total de un proyecto 44](#_Toc427061182)

[Investigar costos de esfuerzo 46](#_Toc427061183)

[Investigar factores que afectan la asignación de precios al software 50](#_Toc427061184)

[Investigar técnicas de estimaciones de costos. 51](#_Toc427061185)

[Investigar modelo COCOMO 56](#_Toc427061186)

[Investigar modelos algorítmicos de costos en la planeación del proyecto 58](#_Toc427061187)

[Investigar duración y personal del proyecto 60](#_Toc427061188)

**SOLUCIÓN**

* 1. **Actividades de Reflexión inicial.**

Identificar conocimientos previos, como estrategia para garantizar la aplicación de los Modelos de calidad en el desarrollo de software: implantación, objetivos organizacionales, técnicas de entrenamiento, definición de planes de mejoramiento (acciones correctivas, preventivas y de mejoramiento), conocimientos previos de procesadores de texto, flujo de información, redacción y ortografía, didáctica, software para presentaciones, Internet, navegadores y otros sistemas y herramientas tecnológicas necesarias.

De acuerdo al ejemplo del cálculo Puntos de Función (PF):

# Realizar la medición a 6 software conocidos, utilizando las imágenes del programa paso a paso.

Debe explicar por cada software de ejemplo lo siguiente:

-Entradas del usuario

-Salidas del usuario

-Peticiones del usuario

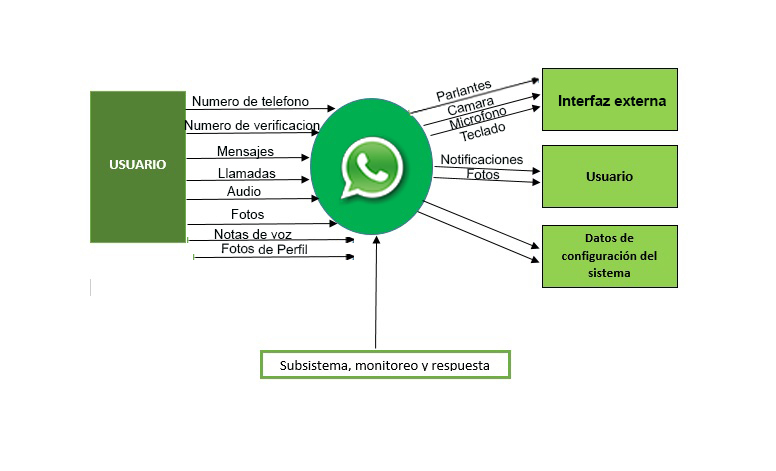
-Archivos

-Interfaces externas

Realizar una tabla explicando cálculo de factores de ajustes.

Finalmente hallar el cálculo Puntos de Función (PF).

**Software 1: WhatsApp**

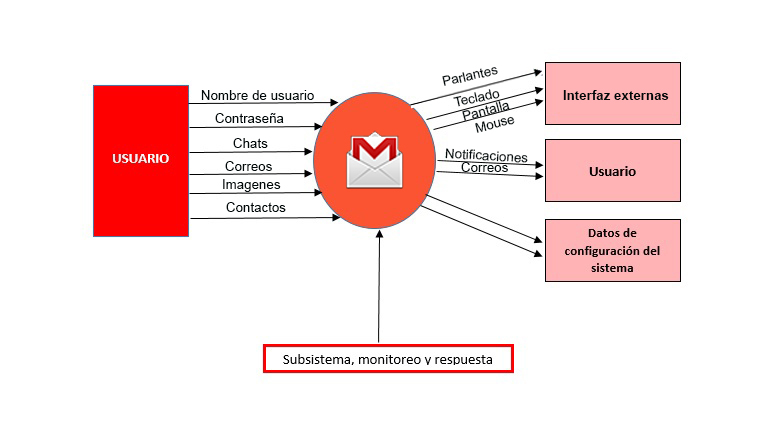
****

**Cálculo Puntos de Función (PF).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Aplicación: WhatsApp** | | | | |
| **Valor dominio de información** | **Cuenta** | **Factor ponderado** | | |
| **Simple** | **Promedio** | **Complejo** |
| **Entradas externas (EE)** | 8 | 3 | 4 | 6 |
| **Salidas externas (SE)** | 2 | 4 | 5 | 7 |
| **Consultas externas (CE)** | 0 | 3 | 4 | 6 |
| **Archivos lógicos internos (ALI)** | 0 | 7 | 10 | 15 |
| **Archivos de interfaz externos (AIE)** | 0 | 5 | 7 | 10 |
| **Conteo total** | 40 | | | |
| **PF = Conteo total x [0,65 + 0,01] x ∑(Fi)** | | | | |
| **Si asumimos ∑(Fi) = 52 , entonces:** | | | | |
| **PF = 40 x [0,65 + 0,01 x 52] = 1372,8** | | | | |

|  |  |  |
| --- | --- | --- |
| **Aplicación: WhatsApp** | | |
| Id | **Calculo de factores de ajuste** | **Calificación (0-5)** |
| 1 | Copias de seguridad y recuperación. | 5 |
| 2 | Comunicación de datos. | 5 |
| 3 | Proceso distribuido. | 4 |
| 4 | Rendimiento crítico. | 4 |
| 5 | Entorno operativo existente y utilizado. | 3 |
| 6 | Entrada de datos interactiva. | 4 |
| 7 | Transacciones de entrada con múltiples pantallas u operaciones. | 3 |
| 8 | Archivos maestros actualizados de forma interactiva. | 3 |
| 9 | Son complejas las entradas, las salidas o las peticiones. | 3 |
| 10 | Es complejo el procesamiento interno. | 4 |
| 11 | Código reutilizable. | 4 |
| 12 | Soporta el sistema múltiples instalaciones en diferentes organizaciones. | 5 |
| 13 | Facilita los cambios y es fácilmente utilizada por el usuario. | 5 |
| Total: 52 | | |

**Software 2: Gmail**

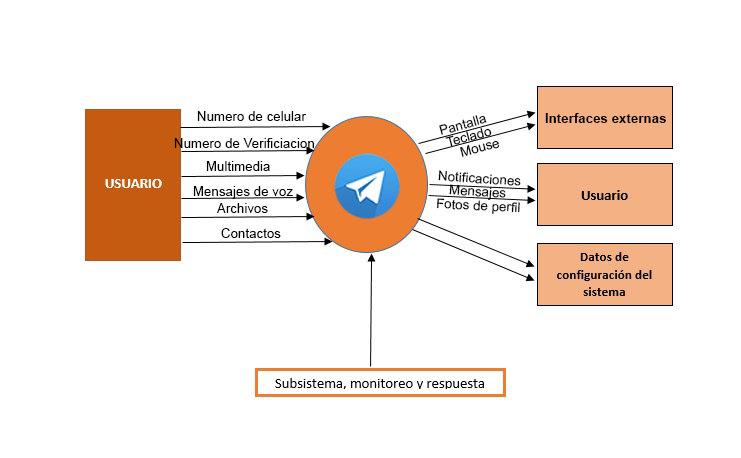
****

**Cálculo Puntos de Función (PF).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Aplicación: Gmail** | | | | |
| **Valor dominio de información** | **Cuenta** | **Factor ponderado** | | |
| **Simple** | **Promedio** | **Complejo** |
| **Entradas externas (EE)** | 6 | 3 | 4 | 6 |
| **Salidas externas (SE)** | 2 | 4 | 5 | 7 |
| **Consultas externas (CE)** | 0 | 3 | 4 | 6 |
| **Archivos lógicos internos (ALI)** | 0 | 7 | 10 | 15 |
| **Archivos de interfaz externos (AIE)** | 4 | 5 | 7 | 10 |
| **Conteo total** |  | | | |
| **PF = Conteo total x [0,65 + 0,01] x ∑(Fi)** | | | | |
| **Si asumimos ∑(Fi) = 50 , entonces:** | | | | |
| **PF = 90 x [0,65 + 0,01 x b] = 2970** | | | | |

|  |  |  |
| --- | --- | --- |
| **Aplicación: Gmail** | | |
| **Id** | **Calculo de factores de ajuste** | **Calificación (0-5)** |
| **1** | Copias de seguridad y recuperación. | 5 |
| **2** | Comunicación de datos. | 5 |
| **3** | Proceso distribuido. | 4 |
| **4** | Rendimiento crítico. | 4 |
| **5** | Entorno operativo existente y utilizado. | 3 |
| **6** | Entrada de datos interactiva. | 4 |
| **7** | Transacciones de entrada con múltiples pantallas u operaciones. | 3 |
| **8** | Archivos maestros actualizados de forma interactiva. | 3 |
| **9** | Son complejas las entradas, las salidas o las peticiones. | 4 |
| **10** | Es complejo el procesamiento interno. | 3 |
| **11** | Código reutilizable. | 3 |
| **12** | Soporta el sistema múltiples instalaciones en diferentes organizaciones. | 4 |
| **13** | Facilita los cambios y es fácilmente utilizada por el usuario. | 5 |
| Total: 50 | | |

**Software 3: Telegram**

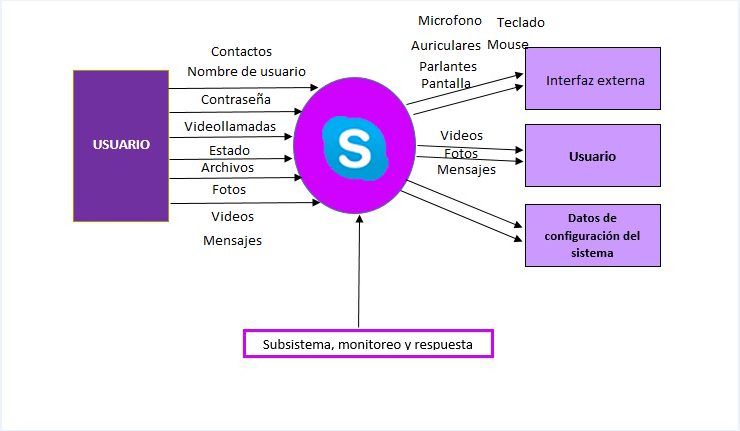
****

**Cálculo Puntos de Función (PF).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Aplicación: Telegram** | | | | |
| **Valor dominio de información** | **Cuenta** | **Factor ponderado** | | |
| **Simple** | **Promedio** | **Complejo** |
| **Entradas externas (EE)** | 6 | 3 | 4 | 6 |
| **Salidas externas (SE)** | 3 | 4 | 5 | 7 |
| **Consultas externas (CE)** | 0 | 3 | 4 | 6 |
| **Archivos lógicos internos (ALI)** | 0 | 7 | 10 | 15 |
| **Archivos de interfaz externos (AIE)** | 3 | 5 | 7 | 10 |
| **Conteo total** | 45 | | | |
| **PF = Conteo total x [0,65 + 0,01] x ∑(Fi)** | | | | |
| **Si asumimos ∑(Fi) = b , entonces:** | | | | |
| **PF = 45 x [0,65 + 0,01 x 48] = 1425,6** | | | | |

|  |  |  |
| --- | --- | --- |
| **Aplicación: Telegram** | | |
| **Id** | **Calculo de factores de ajuste** | **Calificación (0-5)** |
| **1** | Copias de seguridad y recuperación. | 4 |
| **2** | Comunicación de datos. | 5 |
| **3** | Proceso distribuido. | 4 |
| **4** | Rendimiento crítico. | 3 |
| **5** | Entorno operativo existente y utilizado. | 4 |
| **6** | Entrada de datos interactiva. | 4 |
| **7** | Transacciones de entrada con múltiples pantallas u operaciones. | 3 |
| **8** | Archivos maestros actualizados de forma interactiva. | 3 |
| **9** | Son complejas las entradas, las salidas o las peticiones. | 4 |
| **10** | Es complejo el procesamiento interno. | 3 |
| **11** | Código reutilizable. | 3 |
| **12** | Soporta el sistema múltiples instalaciones en diferentes organizaciones. | 3 |
| **13** | Facilita los cambios y es fácilmente utilizada por el usuario. | 5 |
| Total: 48 | | |

**Software 4: Skype**

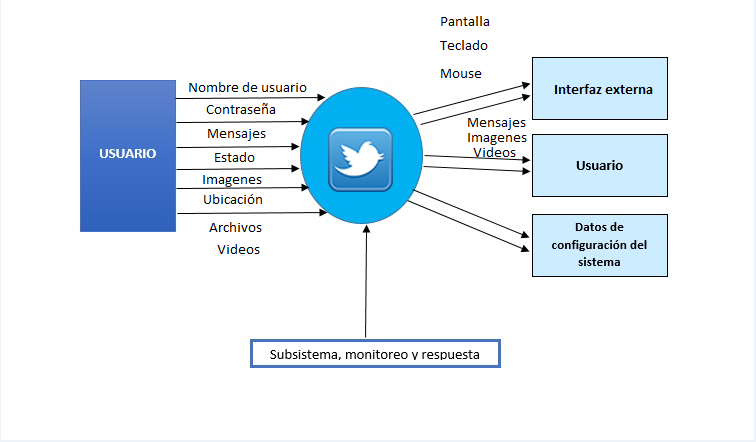
****

**Cálculo Puntos de Función (PF).**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Aplicación: Skype** | | | | | |
| **Valor dominio de información** | **Cuenta** | **Factor ponderado** | | | **Total** |
| **Simple** | **Promedio** | **Complejo** |  |
| **Entradas externas (EE)** | 7 | 3 | 4 | 6 | 42 |
| **Salidas externas (SE)** | 7 | 4 | 5 | 7 | 49 |
| **Consultas externas (CE)** | 4 | 3 | 4 | 6 | 24 |
| **Archivos lógicos internos (ALI)** | 0 | 7 | 10 | 15 | 0 |
| **Archivos de interfaz externos (AIE)** | 3 | 5 | 7 | 10 | 30 |
| **Conteo total** |  |  |  |  | **145** |
| **PF = Conteo total x [0,65 + 0,01] x ∑(Fi)** | | | | |  |
| **Si asumimos ∑(Fi) = 54 , entonces:** | | | | |  |
| **PF = 145 x [0,65 + 0,01x 54] = 5167** | | | | |  |

|  |  |  |
| --- | --- | --- |
| ***Aplicación: Skype*** | | |
| Id | **Calculo de factores de ajuste** | Calificación |
|  | 1. Copias de seguridad y recuperación. | 5 |
|  | 2. Comunicación de datos. | 4 |
|  | 3. Proceso distribuido. | 4 |
|  | 4. Rendimiento crítico. | 3 |
|  | 5. Entorno operativo existente y utilizado. | 3 |
|  | 6. Entrada de datos interactiva. | 4 |
|  | 7. Transacciones de entrada con múltiples pantallas u operaciones. | 5 |
|  | 8. Archivos maestros actualizados de forma interactiva. | 4 |
|  | 9. Son complejas las entradas, las salidas o las peticiones. | 4 |
|  | 10. Es complejo el procesamiento interno. | 5 |
|  | 11. Código reutilizable. | 5 |
|  | 12. Soporta el sistema múltiples instalaciones en diferentes organizaciones. | 4 |
|  | 13 Facilita los cambios y es fácilmente utilizada por el usuario. | 4 |
| **Total: 54** | | |

**Software 5: Twitter**

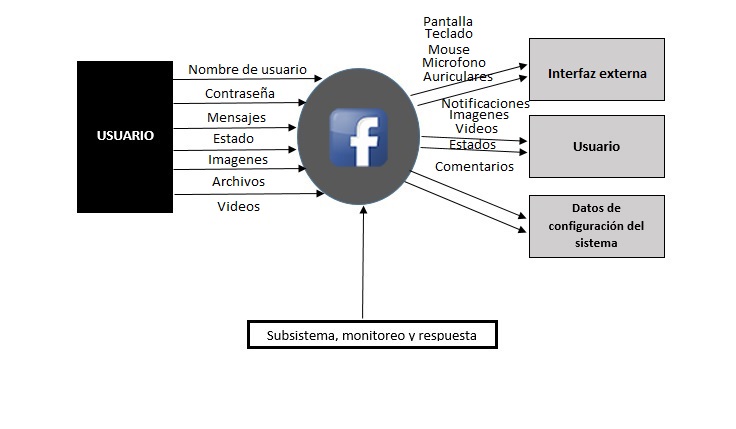
****

**Cálculo Puntos de Función (PF).**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Aplicación: Twitter** | | | | | |
| **Valor dominio de información** | **Cuenta** | **Factor ponderado** | | | **Total** |
| **Simple** | **Promedio** | **Complejo** |
| **Entradas externas (EE)** | 4 | 3 | 4 | 6 | 24 |
| **Salidas externas (SE)** | 3 | 4 | 5 | 7 | 21 |
| **Consultas externas (CE)** | 4 | 3 | 4 | 6 | 24 |
| **Archivos lógicos internos (ALI)** | 0 | 7 | 10 | 15 | 0 |
| **Archivos de interfaz externos (AIE)** | 3 | 5 | 7 | 10 | 30 |
| **Conteo total** |  |  |  |  | **99** |
| **PF = Conteo total x [0,65 + 0,01] x ∑(Fi)** | | | | |  |
| **Si asumimos ∑(Fi) = 59 , entonces:** | | | | |  |
| **PF = 99 x [0,65 + 0,01x 59] = 3855** | | | | |  |

|  |  |  |
| --- | --- | --- |
| ***Aplicación: Twitter*** | | |
| Id | **Calculo de factores de ajuste** | Calificación |
|  | 1. Copias de seguridad y recuperación. | 5 |
|  | 2. Comunicación de datos. | 5 |
|  | 3. Proceso distribuido. | 4 |
|  | 4. Rendimiento crítico. | 5 |
|  | 5. Entorno operativo existente y utilizado. | 4 |
|  | 6. Entrada de datos interactiva. | 5 |
|  | 7. Transacciones de entrada con múltiples pantallas u operaciones. | 5 |
|  | 8. Archivos maestros actualizados de forma interactiva. | 4 |
|  | 9. Son complejas las entradas, las salidas o las peticiones. | 5 |
|  | 10. Es complejo el procesamiento interno. | 4 |
|  | 11. Código reutilizable. | 4 |
|  | 12. Soporta el sistema múltiples instalaciones en diferentes organizaciones. | 4 |
|  | 13 Facilita los cambios y es fácilmente utilizada por el usuario. | 5 |
| **Total: 59** | | |

**Software 6: Facebook**

****

**Cálculo Puntos de Función (PF).**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Aplicación: Facebook** | | | | | |
|  | **Cuenta** | **Factor ponderado** | | | **Total** |
| **Simple** | **Promedio** | **Complejo** |  |
| **Entradas externas (EE)** | 3 | 3 | 4 | 6 | 24 |
| **Salidas externas (SE)** | 5 | 4 | 5 | 7 | 35 |
| **Consultas externas (CE)** | 4 | 3 | 4 | 6 | 24 |
| **Archivos lógicos internos (ALI)** | 0 | 7 | 10 | 15 | 0 |
| **Archivos de interfaz externos (AIE)** | 5 | 5 | 7 | 10 | 50 |
| **Conteo total** |  |  |  |  | **133** |
| **PF = Conteo total x [0,65 + 0,01] x ∑(Fi)** | | | | |  |
| **Si asumimos ∑(Fi) = 56 , entonces:** | | | | |  |
| **PF = 133 x [0,65 + 0,01x56] = 4915** | | | | |  |

|  |  |  |
| --- | --- | --- |
| ***Aplicación: Facebook*** | | |
| Id | **Calculo de factores de ajuste** | Calificación |
|  | 1. Copias de seguridad y recuperación. | 5 |
|  | 2. Comunicación de datos. | 4 |
|  | 3. Proceso distribuido. | 4 |
|  | 4. Rendimiento crítico. | 3 |
|  | 5. Entorno operativo existente y utilizado. | 4 |
|  | 6. Entrada de datos interactiva. | 5 |
|  | 7. Transacciones de entrada con múltiples pantallas u operaciones. | 5 |
|  | 8. Archivos maestros actualizados de forma interactiva. | 4 |
|  | 9. Son complejas las entradas, las salidas o las peticiones. | 5 |
|  | 10. Es complejo el procesamiento interno. | 4 |
|  | 11. Código reutilizable. | 5 |
|  | 12. Soporta el sistema múltiples instalaciones en diferentes organizaciones. | 4 |
|  | 13 Facilita los cambios y es fácilmente utilizada por el usuario. | 4 |
| **Total: 56** | | |

# Investigar métricas de la calidad de software, 2 ejemplos

**Métricas de la calidad de software**

Son un conjunto de reglas generadas para la creación de productos de software con calidad, que si se siguen correctamente pueden garantizar que el proyecto dará como resultado la satisfacción del cliente.

Se usan para poder medir en términos contables la calidad de los procesos en que se realiza dicho producto y evitar errores comunes.

Las métricas del Software comprenden un amplio rango de actividades diversas, estas son algunas:

‣Aseguramiento y control de calidad

‣Modelos de fiabilidad

‣Modelos y evaluación de ejecución

‣Modelos y medidas de productividad

**Ejemplos:**

1. **Métricas de complejidad:** volumen, tamaño, anidaciones, entre otras.
2. **Métricas de competencia:** rapidez, eficiencia, competencia, entre otras.

# Investigar métricas para el esquema de puntuación, 2 ejemplos

**Métricas para el esquema de puntuación**

Las métricas pueden ir en forma de lista de comprobación para evaluar y puntuar atributos específicos del software.

McCall, propuso un esquema de puntuación en una escala del 0 (bajo) al 10 (alto).

**Ejemplos:**

1. **Facilidad de auditoria:** la facilidad con la que se puede comprobar el cumplimiento de los estándares.
2. **Exactitud**: la exactitud de los cálculos y del control.

# Investigar Factores de calidad ISO 9126

**ISO 9126** es un estándar internacional para la evaluación de la calidad del software. Está reemplazado por el proyecto SQuaRE, ISO 25000:2005, el cual sigue los mismos conceptos.

El estándar está dividido en cuatro partes las cuales dirigen, realidad, métricas externas, métricas internas y calidad en las métricas de uso y expendido.

El estándar ISO/IEC 9126 proviene desde el modelo establecido en 1977 por McCall y sus colegas, los cuales propusieron un modelo para especificar la calidad del software. El modelo de calidad McCall está organizado sobre tres tipos de Características de Calidad:

**Factores (especificar):** Describen la visión externa del software, como es visto por los usuarios.

**Criterios (construir):** Describen la visión interna del software, como es visto por el desarrollador.

**Métricas (controlar):** Se definen y se usan para proveer una escala y método para la medida.

El modelo de calidad establecido en la primera parte del estándar, [ISO 9126-1](https://es.wikipedia.org/w/index.php?title=ISO/IEC_9126-1&action=edit&redlink=1), clasifica la calidad del software en un conjunto estructurado de características y subcaracterísticas de la siguiente manera:

* **Funcionalidad**- Un conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen las necesidades implícitas o explícitas.
  + Adecuación - Atributos del software relacionados con la presencia y aptitud de un conjunto de funciones para tareas especificadas.
  + Exactitud - Atributos del software relacionados con la disposición de resultados o efectos correctos o acordados.
  + Interoperabilidad - Atributos del software que se relacionan con su habilidad para la interacción con sistemas especificados.
  + Seguridad - Atributos del software relacionados con su habilidad para prevenir acceso no autorizado ya sea accidental o deliberado, a programas y datos.
  + Cumplimiento funcional.
* **Fiabilidad** - Un conjunto de atributos relacionados con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un período establecido.
  + Madurez - Atributos del software que se relacionan con la frecuencia de falla por fallas en el software.
  + Recuperabilidad - Atributos del software que se relacionan con la capacidad para restablecer su nivel de desempeño y recuperar los datos directamente afectos en caso de falla y en el tiempo y esfuerzo relacionado para ello.
  + Tolerancia a fallos - Atributos del software que se relacionan con su habilidad para mantener un nivel especificado de desempeño en casos de fallas de software o de una infracción a su interfaz especificada.
  + Cumplimiento de Fiabilidad - La capacidad del producto software para adherirse a normas, convenciones o legislación relacionadas con la fiabilidad.
* **Usabilidad** - Un conjunto de atributos relacionados con el esfuerzo necesario para su uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.
  + Aprendizaje- Atributos del software que se relacionan al esfuerzo de los usuarios para reconocer el concepto lógico y sus aplicaciones.
  + Comprensión - Atributos del software que se relacionan al esfuerzo de los usuarios para reconocer el concepto lógico y sus aplicaciones.
  + Operatividad - Atributos del software que se relacionan con el esfuerzo de los usuarios para la operación y control del software.
* **Eficiencia** - Conjunto de atributos relacionados con la relación entre el nivel de desempeño del software y la cantidad de recursos necesitados bajo condiciones establecidas.
  + Comportamiento en el tiempo - Atributos del software que se relacionan con los tiempos de respuesta y procesamiento y en las tasas de rendimientos en desempeñar su función.
  + Comportamiento de recursos - Usar las cantidades y tipos de recursos adecuados cuando el software lleva a cabo su función bajo condiciones determinadas.
* **Mantenibilidad**- Conjunto de atributos relacionados con la facilidad de extender, modificar o corregir errores en un sistema software.
  + Estabilidad - Atributos del software relacionados con el riesgo de efectos inesperados por modificaciones.
  + Facilidad de análisis - Atributos del software relacionados con el esfuerzo necesario para el diagnóstico de deficiencias o causas de fallos, o identificaciones de partes a modificar.
  + Facilidad de cambio - Atributos del software relacionados con el esfuerzo necesario para la modificación, corrección de falla, o cambio de ambiente.
  + Facilidad de pruebas - Atributos del software relacionados con el esfuerzo necesario para validar el software modificado.
* **Portabilidad** - Conjunto de atributos relacionados con la capacidad de un sistema software para ser transferido desde una plataforma a otra.
  + Capacidad de instalación - Atributos del software relacionados con el esfuerzo necesario para instalar el software en un ambiente especificado.
  + Capacidad de reemplazamiento - Atributos del software relacionados con la oportunidad y esfuerzo de usar el software en lugar de otro software especificado en el ambiente de dicho software especificado.
  + Adaptabilidad - Atributos del software relacionados con la oportunidad para su adaptación a diferentes ambientes especificados sin aplicar otras acciones o medios que los proporcionados para este propósito por el software considerado.
  + Co-Existencia - Coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes.

# Investigar estructura para las métricas de software

La medición asigna números o símbolos a atributos de entidades en el mundo real. Para conseguirlo es necesario un modelo de medición que comprenda un conjunto consistente de reglas.

Existe la necesidad de medir y controlar la complejidad del software, es bastante difícil obtener un solo valor para representar una "métrica de calidad", sin embargo es posible desarrollar medidas de diferentes atributos internos del programa como ser: modularidad efectiva, independencia funcional y otros atributos. Estas métricas y medidas obtenidas pueden utilizarse como indicadores independientes de la calidad de los modelos de análisis y diseño.

Los principios básicos de la medición, sugeridos por Roche, pueden caracterizarse mediante cinco actividades:

* **Formulación.** Obtención de medidas y métricas del software apropiadas para la representación del software en cuestión.
* **Colección.** Mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
* **Análisis.** Cálculo de las métricas y la aplicación de herramientas matemáticas.
* **Interpretación.** Evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.
* **Realimentación.** Recomendaciones obtenidas de la interpretación de métricas técnicas transmitidas al equipo software.

La métrica obtenida y las medidas que conducen a ello deberían ser:

* Simple y fácil de calcular.
* Empírica e intuitivamente persuasiva.
* Consistente y objetiva.
* Consistente en el empleo de unidades y tamaños.
* Independiente del lenguaje de programación.
* Un eficaz mecanismo para la realimentación de calidad.

La experiencia indica que una métrica técnica se usa únicamente si es intuitiva y fácil de calcular. Si se requiere docenas de contadores y se han de utilizar complejos cálculos, la métrica no será ampliamente utilizada.

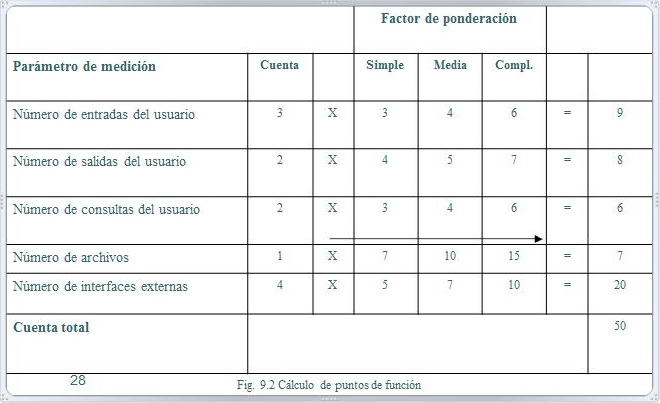
# Investigar métricas del modelo de análisis

En esta fase es deseable que las métricas técnicas proporcionen una visión interna a la calidad del modelo de análisis. Estas métricas examinan el modelo de análisis con la intención de predecir el "tamaño" del sistema resultante; es probable que el tamaño y la complejidad del diseño estén directamente relacionadas.

La métrica del punto de función (PF) se puede utilizar como medio para predecir el tamaño de un sistema obtenido a partir de un modelo de análisis. Para visualizar esta métrica se utiliza un diagrama de flujo de datos, el cual se evaluar para determinar las siguientes medidas clave que son necesarias para el cálculo de la métrica de punto de función:

* Número de entradas del usuario
* Número de salidas del usuario
* Número de consultas del usuario
* Número de archivos
* Número de interfaces externas

La cuenta total debe ajustarse utilizando la siguiente ecuación: PF = cuenta-total x (0,65 + 0,01 x åFi)



Donde cuenta total es la suma de todas las entradas PF obtenidas de la figura 9.2 y Fi (i=1 a 14) son los "valores de ajuste de complejidad"

Para el ejemplo descrito en la figura 9.2 se asume que la EFi es 46 (un producto moderadamente complejo), por consiguiente:

**PF = 50 x (0,65 + 0,01 x 46) = 56**

Basándose en el valor previsto del PF obtenido del modelo de análisis, el equipo del proyecto puede estimar el tamaño global de implementación de las funciones de interacción. Asuma que los datos de los que se dispone indican que un PF supone 60 líneas de código (se utilizará un lenguaje orientado a objetos) y que en un esfuerzo de un mes-persona se producen 12 PF. Estos datos históricos proporcionan al gestor del proyecto una importante información de planificación basada en el modelo de análisis en lugar de estimaciones preliminares.

# Investigar métricas Bang (de marco)

Al igual que la métrica de punto de función, la métrica Bang puede emplearse para desarrollar una indicación del tamaño del software a implementar como consecuencia del modelo de análisis.

Desarrollada por DeMarco, la métrica Bang es «una indicación independiente de la implementación del tamaño del sistema». Para calcular la métrica Bang, el desarrollador de software debe evaluar primero un conjunto de primitivas (elementos del modelo de análisis que no se subdividen más en el nivel de análisis).

Las primitivas se determinan evaluando el modelo de análisis y desarrollando cuentas para los siguientes elementos:

**Primitivas funcionales (PFu):** Transformaciones (burbujas) que aparecen en el nivel inferior de un diagrama de flujo de datos.

**Elementos de datos (ED):** Los atributos de un objeto de datos, los elementos de datos son datos no compuestos y aparecen en el diccionario de datos.

**Objetos (OB):** Objetos de datos.

**Relaciones (RE):** Las conexiones entre objetos de datos.

**Estados (ES):** El número de estados observables por el usuario en el diagrama de transición de estados.

**Transiciones (TR):** El número de transiciones de estado en el diagrama de transición de estados.

Además de las seis primitivas apuntadas arriba, se determinan las cuentas adicionales para:

**Primitivas modificadas de función manual (PMFu):** Funciones que caen fuera del límite del sistema y que deben modificarse para acomodarse al nuevo sistema.

**Elementos de datos de entrada (EDE):** Aquellos elementos de datos que se introducen en el sistema.

**Elementos de datos de salida (EDS):** Aquellos elementos de datos que se sacan del sistema.

**Elementos de datos retenidos (EDR):** Aquellos elementos de datos que son retenidos (almacenados) por el sistema.

**Muestras (tokens) de datos (TCi):** Las muestras de datos (elementos de datos que no se subdividen dentro de una primitiva funcional) que existen en el límite de la i-ésima primitiva funcional (evaluada para cada primitiva).

**Conexiones de relación (REi):** Las relaciones que conectan el i-ésimo objeto en el modelo de datos con otros objetos.

DeMarco sugiere que la mayoría del software se puede asignar a uno de los dos dominios siguientes, dominio de función o dominio de datos, dependiendo de la relación RE/PFu. Las aplicaciones de dominio de

función (encontradas comúnmente en aplicaciones de ingeniería y científicas) hacen hincapié en la transformación de datos y no poseen generalmente estructuras de datos complejas. Las aplicaciones de dominio de datos (encontradas comúnmente en aplicaciones de sistemas de información) tienden a tener modelos de datos complejos.

**RE/PFu < 0,7 implica una aplicación de dominio de función**

**0,8 < RE/PFu < 1,4 indica una aplicación híbrida**

**RE/PFu > 13 implica una aplicación de dominio de datos**

Como diferentes modelos de análisis harán una partición del modelo con mayor o menor grado de refinamiento. DeMarco sugiere que se emplee una cuenta media de muestra (token) por primitiva

**Teavg= DC,IPFu**

para controlar la uniformidad de la partición a través de muchos diferentes modelos dentro del dominio de una aplicación.

Para calcular la métrica Bang para aplicaciones de dominio de función, se emplea el siguiente algoritmo:

**Asignar a bang un valor inicial = 0;**

**Mientras queden primitivas funcionales por evaluar**

**Calcular cuenta-token alrededor del límite de la primitiva i;**

**Calcular el incremento PFu corregido (IPFuC)**

**Asignar la primitiva a una clase**

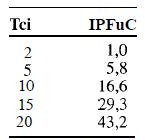
**Evaluar la clase y anotar el peso valorado**

**Multiplicar IPFuC por el peso valorado**

**bang = bang + ponderación IPFuC;**

**FinMientras**

La cuenta-token se calcula determinando cuántos símbolos léxicos (tokens) diferentes son «visibles» dentro de la primitiva. Es posible que el número de símbolos léxicos (tokens) y el número de elementos de datos sea diferente, si los elementos de datos pueden moverse desde la entrada a la salida sin ninguna transformación interna. La IPFuC corregida se determina de una tabla publicada por DeMarco. A continuación, se presenta una versión muy abreviada:



La ponderación valorada apuntada en el algoritmo anterior se calcula de dieciséis clases diferentes de primitivas funcionales definidas por DeMarco. Se asigna una ponderación que va de 0,6 (encaminamiento simple de datos) a 2,5 (funciones de gestión de datos) dependiendo de la clase de la primitiva.

Para aplicaciones de dominio de datos, se calcula la métrica bang mediante el siguiente algoritmo:

**Asignar a bang el valor inicial = 0;**

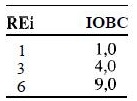
**Mientras queden objetos por evaluar en el modelo de datos**

**Calcular la cuenta de relaciones’ del objeto i**

**Calcular el incremento de OB corregido (IOBC); bang = bang t IOBC;**

**FinMientras**

El IOBC corregido se determina también de una tabla publicada por DeMarco. A continuación se muestra una versión abreviada:



Una vez que se ha calculado la métrica bang, se puede emplear el historial anterior para asociarla con el esfuerzo y el tamaño. DeMarco sugiere que las organizaciones se construyan sus propias versiones de tablas IPFuC e IOBC para calibrar la información de proyectos completos de software.

# Investigar métricas de la calidad de especificación

Davis y sus colegas proponen una lista de características que pueden emplearse para valorar la calidad del modelo de análisis y la correspondiente especificación de requisitos: especificidad (ausencia de ambigüedad), compleción, corrección, comprensión, capacidad de verificación, consistencia interna y externa, capacidad de logro, concisión, trazabilidad, capacidad de modificación, exactitud y capacidad de reutilización. Además, los autores apuntan que las especificaciones de alta calidad deben estar almacenadas electrónicamente, ser ejecutables o al menos interpretables, anotadas por importancia y estabilidad relativas, con su versión correspondiente, organizadas, con referencias cruzadas y especificadas al nivel correcto de detalle.

Aunque muchas de las características anteriores parecen ser de naturaleza cualitativa, Davis sugiere que todas puedan representarse usando una o más métricas. Por ejemplo, asumimos que hay n, requisitos en una especificación, tal como:

http://1.bp.blogspot.com/-2E88u3W6kFE/UK69-ea0fdI/AAAAAAAAAH8/_o2-Br-HWo8/s1600/1.jpg

donde nf es el número de requisitos funcionales y nnf es el número de requisitos no funcionales (por ejemplo, rendimiento).

Para determinar la especificidad (ausencia de ambigüedad) de los requisitos. Davis sugiere una métrica basada en la consistencia de la interpretación de los revisores para cada requisito:

http://1.bp.blogspot.com/-Gyp43nP-qlo/UK6-AuWsfII/AAAAAAAAAIE/3hHepj9d13E/s1600/2.jpg

donde nUi es el número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas. Cuanto más cerca de 1 esté el valor de Q, menor será la ambigüedad de la especificación.

La compleción de los requisitos funcionales puede determinarse calculando la relación:

http://4.bp.blogspot.com/-xAO-TZvdUfc/UK6-ZFCdzcI/AAAAAAAAAIM/2feRXe1Yhjo/s1600/3.jpg

donde u, es el número de requisitos únicos de función, ni es el número de entradas (estímulos) definidos o implicados por la especificación y n, es el número de estados especificados. La relación Q, mide el porcentaje de funciones necesarias que se han especificado para un sistema. Sin embargo, no trata los requisitos no funcionales. Para incorporarlos a una métrica global completa, debemos considerar el grado de validación de los requisitos.

http://3.bp.blogspot.com/-QnyaoOlYbbI/UK6-q8II4XI/AAAAAAAAAIU/nCKYg14aGVY/s1600/4.jpg

donde n, es el número de requisitos que se han validado como correctos y n," el número de requisitos que no se han validado todavía.

# Investigar métricas del modelo de diseño

Las métricas de diseño para el software, como otras métricas del software, no son perfectas. Continúa el debate sobre la eficacia y cómo deberían aplicarse. Muchos expertos argumentan que se necesita más experimentación hasta que se puedan emplear las métricas de diseño. Y sin embargo, el diseño sin medición es una alternativa inaceptable.

**Diseño:** El proceso de aplicar distintas técnicas, herramientas y principios sin el propósito de definir un dispositivo, proceso o sistema; con los suficientes detalles como para permitir su realización.

**Propósito de un diseñador:** Especificar la estructura interna y los detalles de procesamiento de un sistema y proporcionar un ensayo de revisión del por qué fueron timadas las decisiones de diseño.

El diseño del sistema es el proceso de describir, organizar y estructurar los componentes del sistema.

El diseño de más alto nivel también es llamado: diseño general, arquitectónico o conceptual. También es una actividad de modelaje.

El objetivo del diseñador: es producir un modelo o representación del software que se continuara más adelante.

El diseño del software es la primera de tres (3) actividades técnicas:

**1) Diseño 2) Codificación. 3) Prueba.**

**Diseño de Datos:** transforma el modelo del campo de información, creado durante el análisis, en las estructuras de datos que se van a requerir para implementar el software.

**Diseño Arquitectónico:** define las relaciones entre los principales elementos estructurales del programa.

**Diseño Procedimental:** transforma los elementos estructurales en una descripción procedimental del software. Se genera el código fuente y para integrar y validar el software, se llevan a cabo las pruebas.

# Investigar métricas de diseño de alto nivel

Éstas se concentran en las características de la estructura del programa  
dándole énfasis a la estructura arquitectónica y en la eficiencia de los módulos.  
Estas métricas son de caja negra, en el sentido de que no se requiere ningún  
conocimiento del trabajo interno de ningún modo en particular del sistema.

Las métricas de alto nivel nos ayudan a localizar los módulos más complejos y, por lo tanto, aquellos en los que debemos poner especial atención. También es utilizada para saber el número de módulos asignados a cada trabajador

**PRUEBAS DE CAJA NEGRA**

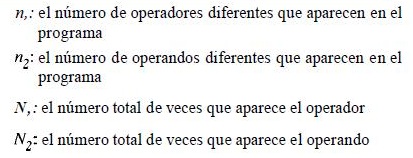
Las pruebas de caja negra, se enfocan en las entradas y salidas del proyecto, sin tener en cuenta su funcionamiento interno, mediante la aplicación de estas pruebas se busca:

1. Probar el desempeño del sistema en su entorno.  
2. Enfocarse en las entradas y salidas, independiente de su funcionamiento interno.  
3. Enfocarse en asegurar que la comunicación entre módulos o componentes del sistema sea acorde a lo especificado.  
4. Pruebas en que se conoce sólo la interfaz.  
5. Se procura ejercitar cada elemento de la interfaz.

# Investigar métricas del código fuente

La teoría de Halstead de la ciencia del software es «probablemente la mejor conocida y más minuciosamente estudiada... medidas compuestas de la complejidad (software)». La ciencia del software propone las primeras «leyes» analíticas para el software de computadora.

La ciencia del software asigna leyes cuantitativas al desarrollo del software de computadora, usando un conjunto de medidas primitivas que pueden obtenerse una vez que se ha generado o estimado el código después de completar el diseño. Estas medidas se listan a continuación:



Halstead usa las medidas primitivas para desarrollar expresiones para la longitud global del programa; volumen mínimo potencial para un algoritmo; el volumen real (número de bits requeridos para especificar un programa); el nivel del programa (una medida de la complejidad del software); nivel del lenguaje (una constante para un lenguaje dado); y otras características tales como esfuerzo de desarrollo, tiempo de desarrollo e incluso el número esperado de fallos en el software.

Halstead expone que la longitud N se puede estimar como:

http://4.bp.blogspot.com/-vo1M-BQG5Gc/UQK7G-ibwzI/AAAAAAAAAKI/5D_d5VoX_J4/s1600/Screenshot+(12h+03m+32s).jpg

y el volumen de programa se puede definir como:

http://3.bp.blogspot.com/-aYaYyIGj-a8/UQK7X202JII/AAAAAAAAAKQ/1RwIzbljGpA/s1600/Screenshot+(12h+04m+56s).jpg

Se debería tener en cuenta que V variará con el lenguaje de programación y representa el volumen de información (en bits) necesarios para especificar un programa.

Teóricamente, debe existir un volumen mínimo para un algoritmo. Halstead define una relación de volumen L como la relación de volumen de la forma más compacta de un programa con respecto al volumen real del programa. Por tanto, L debe ser siempre menor de uno. En términos de medidas primitivas, la relación de volumen se puede expresar como:

http://4.bp.blogspot.com/-q0scoTPwazA/UQK71eFoSMI/AAAAAAAAAKY/7cgIT4jNV7M/s1600/Screenshot+(12h+06m+54s).jpg

# Investigar métricas para la prueba

Aunque se ha escrito mucho sobre las métricas del software para pruebas, la mayoría de las métricas propuestas se concentran en el proceso de prueba, no en las características técnicas de las pruebas mismas. En general, los responsables de las pruebas deben fiarse de las métricas de análisis, diseño y código para que les guíen en el diseño y ejecución de los casos de prueba.

Las métricas basadas en la función pueden emplearse para predecir el esfuerzo global de las pruebas. Se pueden juntar y correlacionar varias características a nivel de proyecto (por ejemplo, esfuerzo y tiempo para las pruebas, errores descubiertos, número de casos de prueba producidos) de proyectos anteriores con el número de PF producidos por un equipo del proyecto. El equipo puede después predecir los valores «esperados» de estas características del proyecto actual.

La métrica bang puede proporcionar una indicación del número de casos de prueba necesarios para examinar las medidas primitivas. El número de primitivas funcionales (PFu), elementos de datos (DE), objetos (OB), relaciones (RE), estados (ES) y transiciones (TR) pueden emplearse para predecir el número y tipos de prueba del software de caja negra y de caja blanca. Por ejemplo, el número de pruebas asociadas con la interfaz hombre-máquina se puede estimar examinando: (1) el número de transiciones (TR) contenidas en la representación estado-transición del IHM y evaluando las pruebas requeridas para ejecutar cada transición, (2) el número de objetos de datos (OB) que se mueven a través de la interfaz y (3) el número de elementos de datos que se introducen o salen.

Las métricas del diseño arquitectónico proporcionan información sobre la facilidad o dificultad asociada con la prueba de integración y la necesidad de software especializado de prueba (por ejemplo, matrices y controladores). La complejidad ciclomática (una métrica de diseño de componentes) se encuentra en el núcleo de las pruebas de caminos básicos. Además, la complejidad ciclomática puede usarse para elegir módulos como candidatos para pruebas más profundas. Los módulos con gran complejidad ciclomática tienen más probabilidad de tendencia a errores que los módulos con menor complejidad ciclomática.

Por esta razón, el analista debería invertir un esfuerzo extra para descubrir errores en el módulo antes de integrarlo en un sistema. Es esfuerzo de las pruebas también se puede estimar usando métricas obtenidas de medidas de Halstead. Usando la definición del volumen de un programa, V, y nivel de programa, NP, el esfuerzo de la ciencia del software, e, puede calcularse como:

http://4.bp.blogspot.com/-PW3YKE4vRjo/UQK-DM-cDUI/AAAAAAAAAKk/vRK0nOiDwqk/s1600/1.jpg

El porcentaje del esfuerzo global de pruebas a asignar a un módulo k se puede estimar usando la siguiente relación:

http://4.bp.blogspot.com/-r9NmxQL8Ljo/UQK-WLwtPoI/AAAAAAAAAKs/5-EtXWUyKLI/s1600/Screenshot+(12h+15m+48s).jpg

donde e (k) se calcula para el módulo k usando la primera ecuaciones y la suma en el denominador de la segunda ecuación es la suma del esfuerzo de la ciencia del software a lo largo de todos los módulos del sistema.

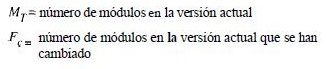
A medida que se van haciendo las pruebas, tres medidas diferentes proporcionan una indicación de la compleción de las pruebas. Una medida de la amplitud de las pruebas proporciona una indicación de cuantos requisitos (del número total de ellos) se han probado.

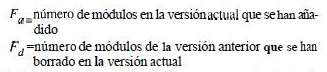
Esto proporciona una indicación de la compleción del plan de pruebas. La profundidad de las pruebas es una medida del porcentaje de los caminos básicos independientes probados en relación al número total de estos caminos en el programa. Se puede calcular una estimación razonablemente exacta del número de caminos básicos sumando la complejidad ciclomática de todos los módulos del programa. Finalmente, a medida que se van haciendo las pruebas y se recogen los datos de los errores, se pueden emplear los perfiles de fallos para dar prioridad y categorizar los errores descubiertos. La prioridad indica la severidad del problema. Las categorías de los fallos proporcionan una descripción de un error, de manera que se puedan llevar a cabo análisis estadísticos de errores.

# Investigar métricas del mantenimiento

Todas las métricas del software pueden usarse para el desarrollo de nuevo software y para el mantenimiento del existente. Sin embargo, se han propuesto métricas diseñadas explícitamente para actividades de mantenimiento.

El estándar EEE 982.1-1988 sugiere un índice de madurez del software (IMS) que proporciona una indicación de la estabilidad de un producto software (basada en los cambios que ocurren con cada versión del producto). Se determina la siguiente información:





El índice de madurez del software se calcula de la siguiente manera:

http://1.bp.blogspot.com/-Ha8zpuqmyZU/UQLA6ZPLX9I/AAAAAAAAALY/BZlnXnuM-w8/s1600/5.jpg

A medida que el IMS se aproxima a 1 ,O el producto se empieza a estabilizar. EL IMS puede emplearse también como métrica para la planificación de las actividades de mantenimiento del software. El tiempo medio para producir una versión de un producto software puede correlacionarse con el IMS desarrollándose modelos empíricos para el mantenimiento.

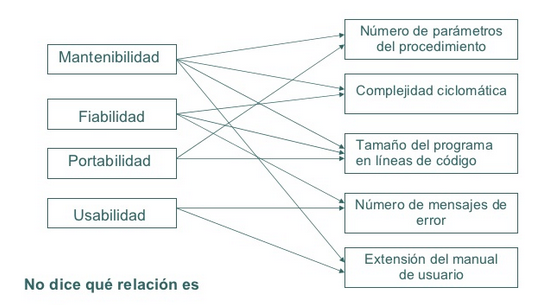
# Investigar métricas para predecir la calidad

A menudo es imposible medir los atributos de calidad del software en forma directa.

Los atributos como la complejidad, la mantenibilidad y la comprensión se ven afectados por diversos factores y no existen métricas directas para ellos.

Más bien es necesario medir un atributo interno del software (como el tamaño) y suponer que existe una relación entre lo que se puede medir y lo que se quiere saber.

De forma ideal, existe una relación clara válida entre los atributos de software internos y externos.



# Investigar métricas del producto

La medición es un elemento clave para cualquier proceso de ingeniería, las medidas se utilizan para comprender de mejor manera los atributos de los modelos que se crean y evaluar  la calidad de los productos de la ingeniería  o de los sistemas que se construyen  
  
La medición es el proceso mediante el cual se asignan números  o símbolos a los atributos de entidades reales para definirlas de acuerdo a reglas claramente establecidas, por supuesto  las medidas por lo general no tienen un alto grado de refinamiento, pero es de gran importancia tratar de medir lo inmedible para comprender de mejor manera entidades particulares del desarrollo de software.  
  
La calidad  es el cumplimiento de los requisitos de funcionalidad  y desempeño explícitamente establecidos, de los estándares de desarrollo  explícitamente documentados y de las características implícitas que se esperan de todo software de desarrollo profesionalmente.



**Factores de calidad de mcall**  
Se dividen en dos grandes grupos:

1. Los que se miden directamente  
2. Los que solo se pueden medir indirectamente

[](http://4.bp.blogspot.com/_hELarvZxWQo/S5bICTCSGvI/AAAAAAAAAG8/iZquzGSVWJY/s1600-h/factoresmcall.JPG)

Estos elementos son cualitativos al igual manera se pretende asignar numero o símbolos a entidades reales y esto merece un modelo de medición que abarque un conjunto de reglas  
  
Una medida proporciona una indicación cuantitativa de la extensión, la cantidad, la dimensión, la capacidad  o el tamaño de algún atributo del producto o proceso.  
  
Medición es el acto de determinar una medida  
  
Métrica es la medida cuantitativa del grado de en qué un sistema, componente o proceso posee un atributo determinado.  
  
Un indicador es una métrica o una combinación de métricas que proporciona conocimiento acerca del proceso de software, un proyecto de software o el propio producto.  
  
Algunos principios en el uso de métricas pueden ser los siguientes.  
  
Una métrica debe tener propiedades matemáticas deseables  
  
Cuando  una métrica representa  una característica de software que aumenta cuando se representan rasgos positivos  o que disminuyen al encontrar rasgos indeseables el valor de la métrica debe aumentar o disminuir en  el mismo sentido.  
  
Cada métrica debe validarse empíricamente en una amplia variedad de contextos antes de publicarse o aplicarse a toma de decisiones.  
  
Existen muchas métricas que cumplen con los principios esto no indica que deban ser totalmente rechazadas o desmerecidas pero hay que tener cuidado al utilizarlas comprender sus objetivos y entender que no pueden ser usadas como comprobación científica sólida.  
  
Las métricas y sus medidas deben ser.  
  
Simples y calculables  
Empírica e intuitivamente persuasiva  
Consistentes y objetivas  
Consistentes en el uso de unidades y dimensiones  
Independientes del lenguaje de programación  
Mecanismo efectivo para la retroalimentación de alta calidad

# Investigar métricas de medición del proceso

Son medidas cuantitativas que permiten a los ingenieros de software obtener una visión de la eficacia del proceso de software y los proyectos que llevan a cabo utilizando el proceso como marco de trabajo.

Los involucrados en analizar, evaluar y recopilar estas métricas son los gestores de software.

Se comienza definiendo un conjunto limitado de medidas del proceso y del proyecto que puedan recopilarse con facilidad, se normaliza empleando métricas orientadas al tamaño o la función.

**Métricas en los dominios del proceso y del proyecto**

Las métricas del proceso se recopilan en el curso de todos los proyectos.

Las métricas del proyecto permiten que un gestor del proyecto de software:

– Valore el estado de un proyecto en curso.

– Rastree los riesgos potenciales.

– Descubra las áreas problema.

– Ajuste el flujo de trabajo o las tareas

– Evalúe la habilidad del equipo del proyecto para controlar la calidad de los productos de trabajo de software.

**Métricas para el proceso y la mejora del proceso de software**

Para mejorar el proceso, se miden sus atributos específicos.

Hay que destacar que el proceso es sólo uno de varios factores controlables en la mejora de la calidad del software y el desempeño organizacional.

# Investigar parámetros involucrados en el costo total de un proyecto

Existen tres parámetros involucrados en el cálculo del costo total de un proyecto de desarrollo de software:

* Los costos de hardware y software, incluyendo el mantenimiento;
* Los costos de viajes y capacitación;
* Los costos de esfuerzo (los costos de pago a los ingenieros de software).

Para muchos proyectos, el costo dominante es el costo del esfuerzo. Las computadoras que son suficientemente poderosas como para desarrollar el software son relativamente baratas. Aunque los costos de viajes pueden ser importantes si un proyecto se desarrolla en sitios distintos, son relativamente bajos para muchos proyectos. Además, el uso de correo electrónico, los chat, fax y teleconferencias reduce los viajes requeridos.

Los costos del esfuerzo no son simplemente los relacionados a los salarios de los ingenieros de software involucrados en el proyecto. Las organizaciones calculan los costos de esfuerzo en función de los costos de sobrecarga donde se toma en cuenta el costo total de hacer funcionar la organización y dividen éste entre el número de personas productivas. Por lo tanto, los siguientes costos son parte de los costos de esfuerzo totales:

1. Los costos de proveer, calentar e iluminar las oficinas;
2. Los costos del personal de apoyo como los contadores, secretarias, limpiadores y técnicos;
3. Los costos de las redes y comunicaciones;
4. Los costos de los recursos centralizados como las bibliotecas, los recursos recreativos, etc.
5. Los costos de seguridad social y de beneficio de empleados como las pensiones y los seguros de salud.

Debido a las consideraciones organizacionales involucradas, asignar precio del proyecto por lo general le concierne al administrador principal de la organización, así como a los administradores del proyecto de software.

El proceso de desarrollo de aplicaciones hipermedia al igual que el de cualquier producto de software, requiere la aplicación de métricas de estimación para garantizar resultados más precisos en su ciclo de vida.

El objetivo de este trabajo es investigar acerca de las técnicas de estimación que existen para los productos hipermedia, presentando métricas para estimación de costos dadas por varios autores de artículos, especialistas en el asunto de aplicaciones hipermedia y web.

# Investigar costos de esfuerzo

Al desarrollar un proyecto se tienen que obtener estimaciones de esfuerzo, así como la duración cronológica del proyecto y el costo. En ocasiones las estimaciones se hacen valiéndose de la experiencia. Si un proyecto pasado funciona de manera adecuada el nuevo proyecto debe de utilizar la misma cantidad de esfuerzo. Y si el proyecto es totalmente distinto entonces la experiencia obtenida, no será suficiente.   
  
Se han desarrollado técnicas de estimación para el desarrollo de software, y cada una tiene sus puntos fuertes y débiles, así como todas tienen en común los siguientes atributos.

1. Se han de establecer de antemano el ámbito del proyecto.
2. Como bases para la realización de estimaciones se usan métricas del software de proyectos pasados.
3. El proyecto se desglosa en partes más pequeñas que se estiman individualmente.

Las medidas directas del proceso de la ingeniería del software incluyen el costo y los esfuerzos aplicados, entre las medidas directas del producto se incluyen: 

* Las líneas de código (LDC) producidas
* Velocidad de ejecución
* Tamaño de memoria
* Los defectos informados durante un período de tiempo establecido.

**Parámetros involucrados en el costo total de un proyecto:**

* Costos de hardware y software incluyendo mantenimiento
* Costos de viajes y capacitación
* Costos de esfuerzo (pago a los ingenieros de Software, en la mayoría de los proyectos es costo dominante es este costo de esfuerzo) a continuación se mostraran algunos costos de esfuerzo.
  + Costos de proveer
  + Costos del personal de apoyo
  + Costos de las redes y las comunicaciones
  + Costos de los recursos centralizados
  + Costos de seguridad social y de beneficio de empleados

**Factores que Afectan la asignación del Costo al software** 

* **Oportunidad de mercado:** penetración de mercado con cotización de bajos costos al inicio
* **Incertidumbre:** si no hay seguridad en el costo estimado por alguna contingencia puede incrementar su costo por encima de lo normal
* **Términos contractuales:** reducir el costo a partir de asumir restricciones como por ejemplo entrega del código fuente y que el desarrollador lo pueda reutilizar en otros proyectos
* **Volatilidad de los requerimientos:** si es probable que los requerimientos cambien, podría reducirse los precios para ganar un contrato. Después del contrato se cargan precios altos a los cambios de requerimientos.
* **Salud financiera:**cuando los desarrolladores tienen dificultades financieras podrían bajar sus costos para ganar contratos. Esto es mejor que quedar fuera del negocio o quebrar.

**Técnicas de Estimación de Costos** No existe una forma simple de hacer una estimación precisa del esfuerzo requerido para desarrollar un sistema de software, por lo general, las estimaciones de los costos del proyecto se cumplen por su propia naturaleza. A pesar de las dificultades e impresiones las organizaciones requieren hacer esfuerzos de software y estimaciones de costos. 

**Modelado Algorítmico de costos**

Se desarrolla un modelo utilizando información histórica de costo que relaciona alguna métrica de software (por lo general su tamaño) con el costo del proyecto. Se hace una estimación de esa métrica y el modelo predice el esfuerzo requerido.

* **Ley de Parkinson**

Establece que el trabajo se extiende para llenar el tiempo disponible. El costo se determina por los recursos disponibles más que por los objetivos logrados. Si el software se tiene que entregar en 12 meses y se dispone de cinco personas el esfuerzo requerido se estima en 60 personas-mes.

* **Estimación por analogía**

Esta técnica es aplicable cuando otros proyectos en el mismo dominio de aplicación se han completado. Se estima el costo de un nuevo proyecto por analogía con estos proyectos completados.

* **Opinión de expertos**

Se consultan varios expertos en las técnicas de desarrollo de software propuestas y en el dominio de la aplicación. Cada uno de ellos estima el costo del proyecto. Estas estimaciones se comparan y discuten. El proceso de estimación se itera hasta que se acuerda una estimación.

* **Asignar precios para ganar**

El costo del software se estima independientemente de lo que el cliente esté dispuesto a pagar por el proyecto. El esfuerzo estimado depende del presupuesto del cliente y no de la funcionalidad del software.

Cada técnica de Estimación tiene sus propias debilidades y fortalezas. Para proyectos grandes se deben aplicar varias técnicas de estimaciones de costos y compara sus resultados. Estas técnicas de estimación son aplicables cuando existe un documento de requerimientos para el sistema.

# Investigar factores que afectan la asignación de precios al software

Los factores internos que afectan la fijación de precios comprenden:

**Factores Internos:**

-  Estrategia global de marketing

-  Objetivos

-  Mezcla de marketing

-  Consideraciones organizacionales

**Factores Externos:**

-  Naturaleza del mercado

-  Demanda

-  Estrategias

-  Precios de los competidores

# Investigar técnicas de estimaciones de costos.

**Técnicas de estimación**

No existe una forma simple de calcular el esfuerzo requerido para desarrollar un sistema informático. Las estimaciones iniciales se hacen bajo la base de la definición de requisitos de usuario de alto nivel.

Pero en una primera etapa del proyecto es difícil producir una estimación precisa de los costos de desarrollo del sistema.

Por ese motivo la estimación se utiliza para definir si el presupuesto del proyecto y el producto se ajustan para que las cifras del presupuesto se cumplan. No se conocen informes de experimentos controlados sobre los costos de los proyectos donde los costos estimados no se utilicen para ajustar el experimento. Un experimento controlado no revelaría la estimación del costo al administrador del proyecto. Los costos reales tendrían que compararse con los estimados del proyecto.

A pesar de esto, las organizaciones necesitan calcular esfuerzos de software y estimaciones de los costos. Para hacerlo, se utilizan una o más de las técnicas descritas en la tabla 1 (Boehm, 1981).

Estos enfoques para la estimación del costo se pueden abordar utilizando enfoque descendente o ascendente. Un enfoque descendente inicia en el nivel de sistema. La estimación comienza examinando la funcionalidad total del producto y cómo es que esa funcionalidad se propaga al interactuar con las subfunciones. Se toman en cuenta los costos de las actividades del nivel del sistema tales como la integración, la administración de la configuración y la documentación.

El enfoque ascendente, en contraste, inicia en el nivel de componente. El sistema se divide en componentes y se calcula el esfuerzo requerido para desarrollar cada uno de éstos. Estos costos se suman para dar el esfuerzo requerido del desarrollo del sistema completo.

Las desventajas del enfoque descendente son las ventajas del ascendente y viceversa. La estimación descendente subestima los costos de resolver problemas técnicos difíciles asociados con componentes específicos como las interfaces para hardware no estándar. No existe justificación detallada de la estimación que se produce. En contraste, la estimación ascendente produce tal justificación y considera cada componente. Sin embargo, este enfoque tiende a subestimar los costos de las actividades del sistema como la integración. La estimación ascendente también es más costosa. Debe haber un diseño inicial del sistema para identificar los componentes a evaluar.

Tabla 1 – Técnicas de estimación de costos

|  |  |
| --- | --- |
| Técnica | Descripción |
| Modelado del algoritmo de costos | Se desarrolla un modelo utilizando información histórica de costos que relaciona alguna métrica de software (por lo general, su tamaño) con el costo del proyecto. Se hace una estimación de esa métrica y el modelo predice el esfuerzo requerido. |
| Opinión de expertos | Se consultan varios expertos en las técnicas de desarrollo de software propuestas y en el [dominio](http://www.monografias.com/trabajos7/doin/doin.shtml) de aplicación. Cada uno de ellos estima el costo del proyecto. Estas estimaciones se comparan y discuten. El proceso de estimación se itera hasta que se acuerda una estimación. |
| Estimación por analogía | Esta técnica es aplicable cuando otros proyectos en el mismo dominio de aplicación se han completado. Se estima el costo de un nuevo proyecto por analogía con estos proyectos completados. Myers (1989) da una [descripción](http://monografias.com/trabajos10/anali/anali.shtml) muy clara de este enfoque. |
| Ley de Parkinson | La [Ley](http://www.monografias.com/trabajos4/leyes/leyes.shtml) de [Parkinson](http://www.monografias.com/trabajos28/parkinsonianos/parkinsonianos.shtml) establece que el trabajo se extiende para llenar el tiempo disponible. El costo se determina por los recursos disponibles más que por los [objetivos](http://www.monografias.com/trabajos16/objetivos-educacion/objetivos-educacion.shtml) logrados. Si el software se tiene que entregar en 12 meses y se dispone de cinco personas, el esfuerzo requerido se estima en 60 personas-mes. |
| Asignar costos para ganar | El costo del software se estima dependiendo de lo que el cliente esté dispuesto a pagar por el proyecto. El esfuerzo estimado depende del presupuesto del cliente y no de la funcionalidad del software. |

Cada técnica de estimación tiene sus propias fortalezas y debilidades. Para proyectos grandes, se deben utilizar varias técnicas de estimación de costos y comparar sus resultados. Si éstos predicen costos radicalmente diferentes, esto indica que no se tiene suficiente información para generar los costos. Se debe buscar más informaciones y repetir el proceso hasta que la estimación converja.

Estas dos técnicas de estimación son aplicables cuando existe un documento de requisitos para el sistema. Éste define a todos los usuarios y los requisitos del sistema. Por lo tanto, se puede generar una estimación razonable de la amplitud de la funcionalidad del sistema a desarrollar. En general, los proyectos grandes de ingeniería de sistemas normalmente tendrán tal documento de requisitos.

Sin embargo, en muchos casos, los costos de muchos proyectos se tienen que estimar utilizando solamente un borrador de solicitudes del usuario del sistema. El análisis y la especificación de requisitos son costosos y los administradores de la compañía necesitan de una estimación inicial del costo del sistema antes de que puedan generar un presupuesto aprobado para este proceso.

Bajo estas circunstancias, "asignar costos para ganar" es una estrategia utilizada comúnmente. La noción de "asignar precios para ganar" parece no ser ética y poco apropiada para los negocios. Sin embargo, tiene algunas ventajas. Un costo del proyecto se acuerda en base a un borrador de la propuesta. Entonces, las negociaciones se llevan a cabo con el cliente para determinar una especificación detallada del proyecto. Esta especificación se restringe por el costo acordado. El comprador y el vendedor deben acordar la funcionalidad aceptable del sistema. El factor clave en muchos proyectos no son los requisitos del proyecto, sino el costo. Los requisitos se cambian de tal forma que no se exceda el costo.

Cuando se estiman los costos de software, los administradores deben tomar en cuenta que existen diferencias importantes entre los proyectos anteriores y los futuros. Una gran variedad de nuevos métodos y técnicas de desarrollo han aparecido en los últimos diez años. Muchos administradores tienen poca experiencia en estas técnicas o poco conocimiento de cómo éstas afectan los costos de los proyectos. Algunos ejemplos de los cambios que afectan las estimaciones de acuerdo con la experiencia son:

* Desarrollo orientado a objetos en lugar de desarrollo orientado a funciones;
* Sistemas cliente-servidor en lugar de sistemas basados en "mainframes";
* Utilización de componentes de software comerciales en lugar de desarrollo de componentes;
* Desarrollar y reutilizar en lugar de desarrollar todas las partes del sistema;
* La utilización de herramientas CASE (Ingeniería de Software Asistida por Computadora) y generadores de programas en lugar de desarrollar software sin ayuda.

Todos estos factores hacen más difícil que los administradores produzcan estimaciones precisas de los costos de producción del software. Su experiencia previa no es relevante para ayudarles a estimar los costos del proyecto de software.

# Investigar modelo COCOMO

El **Modelo Constructivo de Costos** (o **COCOMO**, por su acrónimo del inglés **CO**nstructive **CO**st **MO**del) es un modelo matemático de base empírica utilizado para estimación de costos[1](https://es.wikipedia.org/wiki/COCOMO#cite_note-1) de software. Incluye tres submodelos, cada uno ofrece un nivel de detalle y aproximación, cada vez mayor, a medida que avanza el proceso de desarrollo del software: básico, intermedio y detallado.

Este modelo fue desarrollado por Barry W. Boehm a finales de los años 70 y comienzos de los 80, exponiéndolo detalladamente en su libro "Software Engineering Economics" (Prentice-Hall, 1981).

**Características:**

Pertenece a la categoría de modelos de subestimaciones basados en estimaciones matemáticas. Está orientado a la magnitud del producto final, midiendo el "tamaño" del proyecto, en líneas de código principalmente.

**Inconvenientes:**

* Los resultados no son proporcionales a las tareas de gestión ya que no tiene en cuenta los recursos necesarios para realizarlas.
* Se puede desviar de la realidad si se indica mal el porcentaje de líneas de comentarios en el código fuente.
* Es un tanto subjetivo, puesto que está basado en estimaciones y parámetros que pueden ser "vistos" de distinta manera por distintos analistas que usen el método.
* Se miden los costes del producto, de acuerdo a su tamaño y otras características, pero no la productividad.
* La medición por líneas de código no es válida para orientación a objetos.
* Utilizar este modelo puede resultar un poco complicado, en comparación con otros métodos (que también sólo estiman).

**Modelos de estimación**

Las ecuaciones que se utilizan en los tres modelos son:[2](https://es.wikipedia.org/wiki/COCOMO#cite_note-Vasco-2)

* E = a(Kl)^b*m(X), en persona-mes
* Tdev = c(E)^d, en meses
* P = E/Tdev, en personas

donde:

* *E* es el esfuerzo requerido por el proyecto, en persona-mes
* *Tdev* es el tiempo requerido por el proyecto, en meses
* *P* es el número de personas requerido por el proyecto
* *a*, *b*, *c* y *d* son constantes con valores definidos en una tabla, según cada submodelo
* *Kl* es la cantidad de líneas de código, en miles.
* *m(X)* Es un multiplicador que depende de 15 atributos.

A la vez, cada submodelo también se divide en **modos** que representan el tipo de proyecto, y puede ser:

* ***Modo orgánico***: un pequeño grupo de programadores experimentados desarrollan software en un entorno familiar. El tamaño del software varía desde unos pocos miles de líneas (tamaño pequeño) a unas decenas de miles (medio).
* ***Modo semilibre*** o ***semiencajado***: corresponde a un esquema intermedio entre el orgánico y el rígido; el grupo de desarrollo puede incluir una mezcla de personas experimentadas y no experimentadas.
* ***Modo rígido*** o ***empotrado***: el proyecto tiene fuertes restricciones, que pueden estar relacionadas con la funcionalidad y/o pueden ser técnicas. El problema a resolver es único y es difícil basarse en la experiencia, puesto que puede no haberla.

# Investigar modelos algorítmicos de costos en la planeación del proyecto

Se basan en fórmulas y parámetros con los que realizan las estimaciones. Son también muy conocidos y empleados en la actualidad, encontrándose en este grupo el modelo COCOMO o puntos de función.

Este enfoque pretende escoger el mejor conjunto de características de los proyectos más cercanos al que se quiere examinar, y realizar las estimaciones con ellas. Para seleccionar dicho grupo se deben calcular los errores producidos al estimar con sus componentes, y buscar aquel conjunto que minimice este error en la estimación.

En principio se podrían obtener todos los posibles conjuntos de características y seleccionar aquel que produzca el menor error en la precisión. Esto resulta fácil para proyectos con pocos factores de esfuerzo, pero por desgracia esta no es la situación más normal. Es frecuente que contengan multitud de características que hacen demasiado ineficiente la aplicación de un algoritmo de fuerza bruta. A este problema se le han buscado muchas soluciones, siendo la más apropiada el uso de algoritmos genéticos.

Un algoritmo genético apropiado para esta selección estaría formado por individuos que representan conjuntos de características de proyectos ya completados. Cada ejemplar tendría asociado un valor de error en la estimación, de manera que con los operadores genéticos y de mutación se sucederían las generaciones, buscando siempre el individuo con menor error.

Una vez identificado, se estimaría el proyecto en cuestión sumándole al valor de esfuerzo del proyecto más parecido el error del individuo más fuerte de la población. Con esto hemos conseguido identificar el conjunto de características más apropiado para tener en cuenta a la hora de estimar, minimizando el error cometido [CHI06].

Este tipo de ajuste de las estimaciones es costoso en términos de computación y tiempo, y no ha sido empleado en la alternativa propuesta en este proyecto porque pensamos que con algoritmos y técnicas más simples se pueden alcanzar resultados equiparables. Por ello, se ha implementado una técnica más eficiente y rápida, basada en un fenómeno estadístico explicado a continuación.

# Investigar duración y personal del proyecto

Es el tiempo de desarrollo del proyecto.

La duración prevista del proyecto y la requerida por el plan del proyecto no son necesariamente la misma.

El tiempo requerido para completar el proyecto está en función del esfuerzo total requerido y no del número de ingenieros que trabajan en él.

**REFERENCIAS**

<http://ldc.usb.ve/~abianc/materias/ci4712/metricas.pdf>

<http://es.slideshare.net/lggb/metricas-de-calidad>

<http://datateca.unad.edu.co/contenidos/301404/301404_ContenidoEnLinea/leccin_44__mtricas_tcnicas_del_software.html>

<http://myslide.es/documents/estructura-para-las-metricas-del-software.html>

<http://ing-software3.blogspot.com/2012/11/metricas-del-modelo-de-analisis.html>

**IMAGENES**

<http://elchrboy.blogspot.com/2010/03/metricas-del-producto-para-el-software.html>

<http://www.monografias.com/trabajos15/estimacion-hipermedia/estimacion-hipermedia.shtml>

<http://clases3gingsof.wikifoundry.com/page/Estimaci%C3%B3n+de+Esfuerzo+y+Costo>

<https://es.wikipedia.org/wiki/COCOMO>