MÉTRICAS Y CONFIABILIDAD DEL SOFTWARE

Dra. Linda Rosenberg
Unisys/NASA GSFC
Bld 6 Código 300.1
Greenbelt, MD 20771
EE. UU.
301-286-0087
Linda.Rosenberg@gsfc.nasa.gov

Martillo
GSFC de la NASA
Edificio 6 Código 302
Greenbelt, MD 20771
EE.UU
301-286-7475
thammer@pop300.gsfc.nasa.gobierno

Jack Shaw
GSFC de la NASA
Edificio 6 Código 302
Greenbelt, MD 20771
EE.UU
301-286-7123
jjshaw@pop300.gsfc.nasa.gov

Abstracto

El IEEE define la confiabilidad como “La capacidad de un sistema o componente para realizar su funciona bajo condiciones establecidas por un período de tiempo específico”. Para la mayoría de proyectos y software gerentes de desarrollo, la confiabilidad se equipara a la corrección, es decir, miran a las pruebas y la número de "errores" encontrados y corregidos. Si bien encontrar y corregir errores descubiertos en las pruebas es necesario para asegurar la confiabilidad, una mejor manera es desarrollar un producto robusto y de alta calidad a través de todas las etapas del ciclo de vida del software. Es decir, la confiabilidad del código entregado está relacionada a la calidad de todos los procesos y productos de desarrollo de software; los requisitos documentación, el código, los planes de prueba y las pruebas.

La confiabilidad del software no está tan bien definida como la confiabilidad del hardware, pero Software Assurance El Centro de Tecnología (SATC) de la NASA se esfuerza por identificar y aplicar métricas al software productos que promuevan y evalúen la confiabilidad. Este artículo analiza cómo la NASA proyecta, en junto con el SATC, están aplicando métricas de software para mejorar la calidad y confiabilidad de productos de software. La confiabilidad es un subproducto de la calidad, y la calidad del software puede ser Medido. Demostraremos cómo estas métricas de calidad ayudan en la evaluación del software. fiabilidad. Concluimos con una breve discusión de las métricas que aplica el SATC para evaluar la confiabilidad.

I. Definiciones

El software no se puede ver ni tocar, pero es esencial para el uso exitoso de las computadoras. Es necesario que se mida y evalúe la fiabilidad del software, como se hace en el hardware. IEEE 610.12-1990 define la confiabilidad como “La capacidad de un sistema o componente para realizar su funciones requeridas bajo condiciones establecidas por un período de tiempo específico”. IEEE 982.1-1988 define la Gestión de la Confiabilidad del Software como “El proceso de optimizar la confiabilidad de software a través de un programa que enfatiza la prevención de errores de software, detección de fallas y eliminación y el uso de mediciones para maximizar la confiabilidad a la luz de las limitaciones del proyecto, tales como como recursos, cronograma y desempeño”. Usando estas definiciones, la confiabilidad del software es compuesto por tres actividades:

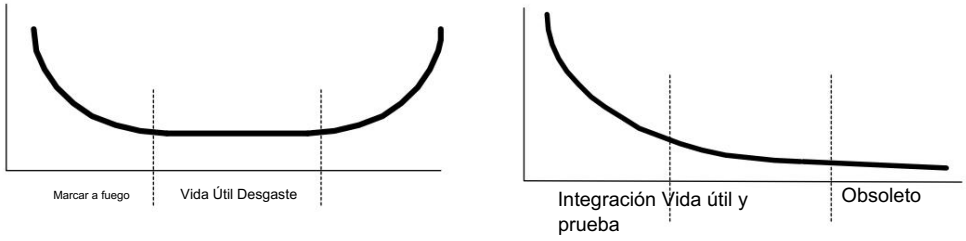
1. Prevención de errores
2. Detección y eliminación de fallas
3. Mediciones para maximizar la confiabilidad, específicamente medidas que respaldan las dos primeras actividades

Ha habido un extenso trabajo en la medición de la confiabilidad utilizando el tiempo medio entre fallas y tiempo medio hasta el fallo.[1] Se ha realizado un modelado exitoso para predecir las tasas de error y

confiabilidad.[1,2,3] Estas actividades abordan el primer y tercer aspecto de la confiabilidad, identificando y eliminando fallas para que el software funcione como se espera con la confiabilidad especificada. Estas medidas se han aplicado con éxito tanto al software como al hardware. Pero en este documento, nos gustaría adoptar un enfoque diferente a la confiabilidad del software, uno que aborde el segundo aspecto de la confiabilidad, la prevención de errores.

II. Errores, fallas y fallas Los términos errores, fallas y
fallas a menudo se usan indistintamente, pero tienen diferentes significados. En el software, un error suele ser una acción
u omisión del programador que da como resultado una falla.
Una falla es un defecto de software que causa una falla, y una falla es la desviación inaceptable de la operación de un programa de los requisitos del programa. Cuando medimos la confiabilidad, por lo general solo medimos los defectos encontrados y los defectos reparados.[4] Si el objetivo es medir completamente la confiabilidad, debemos abordar la prevención e investigar el desarrollo a partir de la fase de requisitos: para qué se desarrollan los programas.

Es importante reconocer que existe una diferencia entre la tasa de fallas de hardware y la tasa de fallas de software. Para el hardware, como se muestra en la Figura 1, cuando el componente se fabrica por primera vez, el número inicial de fallas es alto pero luego disminuye a medida que se identifican y eliminan los componentes defectuosos o se estabilizan. Luego, el componente ingresa a la fase de vida útil, donde se encuentran pocas fallas, si es que se encuentran algunas. A medida que el componente se desgasta físicamente, la tasa de fallas comienza a aumentar.[1]



Tasa de fallas de hardware Tasa de fallas de software

Figura 1: Tasas de falla

Sin embargo, el software tiene una tasa diferente de identificación de fallas o errores. Para el software, la tasa de error está en el nivel más alto en la integración y la prueba. A medida que se prueba, los errores se identifican y eliminan. Esta eliminación continúa a un ritmo más lento durante su uso operativo; el número de errores disminuye continuamente, asumiendo que no se introducen nuevos errores. El software no tiene partes móviles y no se desgasta físicamente como el hardware, pero deja de ser útil y se vuelve obsoleto.[5]

Para aumentar la confiabilidad mediante la prevención de errores de software, el enfoque debe estar en los requisitos integrales y un plan de prueba integral, asegurando que se prueben todos los requisitos. El enfoque también debe estar en la capacidad de mantenimiento del software, ya que habrá una fase de "vida útil" en la que se necesitará ingeniería de mantenimiento. Por lo tanto, para prevenir errores de software, debemos:

1. Comience con los requisitos, asegurándose de que el producto desarrollado sea el especificado, que todos los requisitos especifiquen de manera clara y precisa la funcionalidad del producto final.
2. Asegúrese de que el código pueda respaldar fácilmente la ingeniería de sostenimiento sin infundir información adicional errores

3. Se cuenta con un programa de prueba integral que verifica toda la funcionalidad establecida en los requisitos. incluido.

La confiabilidad como atributo de calidad Existen
muchos modelos diferentes para la calidad del software, pero en casi todos los modelos, la confiabilidad es uno de los criterios, atributos o características que se incorporan. ISO 9126 [1991] define seis características de calidad, una de las cuales es la confiabilidad. IEEE Std 982.2-1988 establece que "un programa de administración de confiabilidad de software requiere el establecimiento de un conjunto equilibrado de objetivos de calidad del usuario y la identificación de objetivos de calidad intermedios que ayudarán a lograr los objetivos de calidad del usuario". [6] Dado que la confiabilidad es un atributo de la calidad, se puede concluir que la confiabilidad del software depende de un software de alta calidad.

La construcción de software de alta confiabilidad depende de la aplicación de atributos de calidad en cada fase del ciclo de vida de desarrollo con énfasis en la prevención de errores, especialmente en las primeras fases del ciclo de vida. Se necesitan métricas en cada fase de desarrollo para medir los atributos de calidad aplicables. IEEE Std 982.2-1988 incluye el diagrama en la Figura 2, que indica la relación de confiabilidad con las diferentes fases del ciclo de vida.[7]

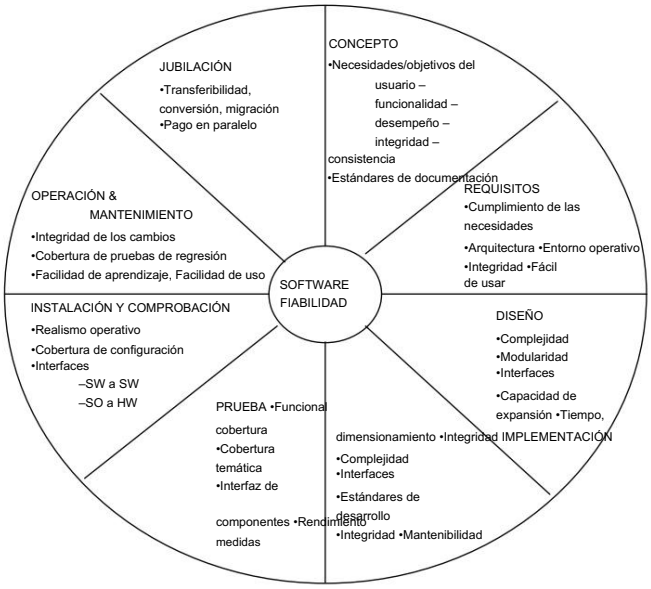


Figura 2: Factores de calidad que afectan la confiabilidad

Al centrarnos en la prevención de errores para la confiabilidad, necesitamos identificar y medir los atributos de calidad aplicables en las diferentes fases del ciclo de vida. Como se discutió anteriormente, debemos centrarnos específicamente en los requisitos, el diseño, la implementación y las fases de prueba.

IV. Métricas de software para confiabilidad Las métricas
de software están siendo utilizadas por el Centro de tecnología de garantía de software (SATC) de la NASA para ayudar a mejorar la confiabilidad mediante la identificación de áreas de la especificación de requisitos de software y el código que pueden causar errores. El SATC también examina el plan de prueba para una cobertura completa de requisitos sin pruebas excesivas (y costosas). El resto de este documento analiza las métricas utilizadas por el SATC para identificar errores potenciales antes de que el software sea

liberado. Los datos del proyecto de la NASA se utilizan para demostrar la aplicación de las métricas. Abordaremos tres fases del ciclo de vida donde se pueden aplicar técnicas de prevención de errores y métricas de software para impactar la confiabilidad: requisitos, codificación y pruebas.

A. Requisitos Métricas de confiabilidad Los

requisitos especifican la funcionalidad que debe incluirse en el software final. Es fundamental que los requisitos se escriban de manera que no haya malentendidos entre el desarrollador y el cliente. Usando los atributos de calidad para la confiabilidad que se muestran en la Figura 2, para el software de alta confiabilidad, los requisitos deben estar estructurados, completos y fáciles de aplicar.

Existen tres formatos principales para la estructura de especificación de requisitos, por IEEE, DOD y NASA.[7,8,9] Estos especifican el contenido de la especificación de requisitos fuera de los requisitos mismos. El uso constante de un formato como estos garantiza que no se omita información crítica, como el entorno operativo.

Los requisitos completos son estables y completos, especificados con suficiente detalle para permitir que el diseño y la implementación continúen. Las especificaciones de requisitos no deben contener frases como TBD (por determinar) o TBA (por agregar) ya que la falta de especificidad de estas frases puede tener un impacto negativo en el diseño, causando una arquitectura inconexa.

Para aumentar la facilidad de uso de los requisitos, generalmente se escriben en inglés [a diferencia de un estilo de escritura especializado (p. ej., notación Z)], lo que puede producir fácilmente una terminología ambigua. Para desarrollar software confiable desde la fase de requisitos en adelante, los requisitos no deben contener términos ambiguos ni terminología que pueda interpretarse como un requisito opcional. Los requisitos ambiguos son aquellos que pueden tener múltiples significados o aquellos que parecen dejar al desarrollador la decisión de implementar o no un requisito.

La importancia de documentar correctamente los requisitos ha provocado que la industria del software produzca un número significativo de ayudas para la creación y gestión de documentos de especificación de requisitos y declaraciones de especificación individuales. Sin embargo, muy pocas de estas ayudas ayudan a evaluar la calidad del documento de requisitos o las propias declaraciones de especificación individuales. El SATC ha desarrollado una herramienta para analizar documentos de requisitos. El software de Medición de requisitos automatizados (ARM)1 se desarrolló para escanear un archivo que contiene el texto de la especificación de requisitos. Durante este proceso de escaneo, busca en cada línea de texto palabras y frases específicas. Estos argumentos de búsqueda (palabras y frases específicas) son indicados por los estudios del SATC como un indicador de la calidad del documento como especificación de requisitos. ARM se ha aplicado a 56 documentos de requisitos de la NASA. Se desarrollaron siete medidas, como se muestra a continuación.

1. Líneas de texto: líneas físicas de texto como medida de tamaño.
2. Imperativos - Palabras y frases que ordenan que se debe hacer o proporcionar algo.
   El número de imperativos se utiliza como base para contar los requisitos.
3. Continuidades: frases que siguen a un imperativo e introducen la especificación de requisitos en
   un nivel inferior, para un recuento de requisitos complementarios.
4. Directrices: referencias proporcionadas a figuras, tablas o notas.

[1] ARM está disponible de forma gratuita en la página de inicio de SATC: http://satc.gsfc.nasa.gov

5. Frases débiles - Cláusulas que pueden causar incertidumbre y dejan espacio para múltiples
interpretación medida de la ambigüedad.
6. Incompleto: declaraciones dentro del documento que tienen TBD (Por determinar) o TBS
(Ser suplido).
7. Opciones: palabras que parecen dar libertad al desarrollador para satisfacer las especificaciones, pero
puede ser ambiguo.

Debe enfatizarse que la herramienta no intenta evaluar la exactitud de los
requisitos especificados. Evalúa declaraciones de especificaciones individuales y el vocabulario utilizado.
indicar los requisitos; también tiene la capacidad de evaluar la estructura del requisito
documento. [10]

Los resultados del análisis de los requisitos de Diseño Nivel 3 y Detallado Nivel 4 son
se muestra en la Tabla 1 con la comparación con otros documentos de la NASA.

Promedio	4772	682	423	49	70	25	63
Nivel 3	1011	588	577	10	242	1	5
Nivel 4	1432	917	289	9	393	2	2

Tabla 1: Análisis de requisitos textuales

Nos preocupa especialmente el número de frases débiles ya que el contrato se licita utilizando
El nivel de diseño 3 y las pruebas de aceptación estarán en contra de estos requisitos. También es motivo de preocupación
que el número de frases débiles ha aumentado en el Nivel Detallado 4, los requisitos utilizados para
escribir el diseño y el código y usarlo en las pruebas de integración.

ARM también evalúa la estructura del documento identificando el número de requisitos en
cada nivel de la estructura de numeración jerárquica. Esta información puede servir como un indicador de
una posible falta de estructura.[10] La falta de estructura afecta la confiabilidad al aumentar la dificultad
al hacer cambios y el posible nivel de detalle inapropiado puede restringir artificialmente
porciones de diseño.

B. Métricas de confiabilidad de diseño y código

Aunque existen lenguajes y formatos de diseño, estos no se prestan a un proceso automatizado.
evaluación y recopilación de métricas. El SATC analiza el código para la estructura y arquitectura.
para identificar posibles módulos propensos a errores en función de la complejidad, el tamaño y la modularidad.

Generalmente se acepta que los módulos más complejos son más difíciles de entender y tienen un
mayor probabilidad de defectos que los módulos menos complejos.[5] Por lo tanto, la complejidad tiene un impacto directo

en la calidad general y específicamente en la mantenibilidad. Si bien hay muchos tipos diferentes de medidas de complejidad, la utilizada por el SATC es la complejidad lógica (ciclomática), que se calcula como el número de rutas de prueba linealmente independientes.

El tamaño es una de las formas más antiguas y comunes de medición de software. El tamaño de los módulos es en sí mismo un indicador de calidad. El tamaño se puede medir por: total de líneas de código, contando todas las líneas; sin comentarios no en blanco que disminuye el total de líneas por el número de espacios en blanco y comentarios; y sentencias ejecutables definidas por un delimitador dependiente del idioma.

El SATC ha encontrado que la evaluación más efectiva es una combinación de tamaño y complejidad. Los módulos con alta complejidad y gran tamaño tienden a tener la confiabilidad más baja.
Los módulos de bajo tamaño y alta complejidad también son un riesgo de confiabilidad porque tienden a ser un código muy conciso, que es difícil de cambiar o modificar.[11]

Aunque estas métricas también son aplicables al código orientado a objetos, se necesitan métricas adicionales para evaluar la calidad de la estructura orientada a objetos. El SATC utiliza las siguientes métricas para el análisis de calidad orientado a objetos: Métodos ponderados por clase (WMC), Respuesta para una clase (RFC), Acoplamiento entre objetos (CBO), Profundidad en el árbol (DIT) y Número de hijos (NOC). Dado que existen muy pocas pautas de la industria para estas métricas, el SATC ha desarrollado pautas basadas en datos de la NASA. Un ejemplo son los métodos ponderados por clase.

Métodos ponderados por clase es un predictor de cuánto tiempo y esfuerzo se requiere para desarrollar y mantener la clase. Cuanto mayor sea el WMC, más pruebas necesarias y mayor mantenimiento. En general, las clases deben tener menos de 20 métodos, pero se aceptan hasta 40. La complejidad de un método no debe exceder 5. Por lo tanto, se prefiere un WMC inferior a 100 y no debe exceder 200. Las clases con un WMC más alto requerirán pruebas exhaustivas para una cobertura integral. También serán difíciles de mantener. Aunque se necesitan más análisis y métricas, podemos concluir con información adicional que estas clases tienen una baja confiabilidad.[12]

C. Pruebas de métricas de confiabilidad

Las métricas de prueba deben adoptar dos enfoques para evaluar exhaustivamente la confiabilidad. El primer enfoque es la evaluación del plan de prueba, asegurando que el sistema contenga la funcionalidad especificada en los requisitos. Esta actividad debería reducir el número de errores debido a la falta de la funcionalidad esperada. El segundo enfoque, comúnmente asociado con la confiabilidad, es la evaluación de la cantidad de errores en el código y la tasa de detección/corrección de los mismos. El SATC ha desarrollado un modelo para simular la detección de errores y proyecta el número de errores restantes y cuándo se identificarán todos.

Para garantizar que el sistema contenga la funcionalidad especificada, se escriben planes de prueba que contienen múltiples casos de prueba; cada caso de prueba se basa en un estado del sistema y prueba algunas funciones que se basan en un conjunto relacionado de requisitos. El objetivo de un programa de verificación efectivo es garantizar que se pruebe cada requisito, lo que implica que si el sistema pasa la prueba, la funcionalidad del requisito se incluirá en el sistema entregado. Se necesita una evaluación de la trazabilidad de los requisitos para los casos de prueba.

En el conjunto total de casos de prueba, cada requisito debe probarse al menos una vez, y algunos requisitos se probarán varias veces porque están involucrados en múltiples estados del sistema en diferentes escenarios y de diferentes maneras. Pero como siempre, el tiempo y la financiación son problemas; Si bien cada requisito debe probarse exhaustivamente, el tiempo limitado y el presupuesto limitado siempre son restricciones para escribir y ejecutar casos de prueba. Es importante asegurarse de que cada requisito se pruebe adecuadamente, pero no en exceso. En algunos casos, los requisitos se pueden agrupar utilizando la criticidad para el éxito de la misión como hilo conductor; estos deben ser ampliamente probados. En otros casos, los requisitos pueden identificarse como de baja criticidad; si ocurre un problema, su funcionalidad no afecta el éxito de la misión mientras se logran pruebas exitosas.[13]

Desde una perspectiva de tendencias de error, el modelo desarrollado por SATC2 emplea una implementación sencilla del modelo Musa para calcular una aproximación no lineal a los errores acumulados encontrados hasta la fecha. Sin embargo, en lugar de utilizar el enfoque probabilístico para determinar los parámetros del modelo, aplicamos un proceso "iterativo" que emplea una técnica de optimización no lineal para ajustar la curva de distribución de error acumulativo generada por el modelo a la curva de distribución de error acumulativo real. Se utiliza una suma de diferencias al cuadrado para determinar el "mejor ajuste". El uso de los valores calculados y otras funciones definidas por Musa nos permite estimar: (a) la cantidad de errores restantes en el producto y (b) el tiempo necesario para detectar los errores restantes.[1]

Para relacionar la utilización de la mano de obra con la detección de los errores que quedan en el producto, usamos una versión modificada del modelo de Musa donde el término lineal en la exponencial se reemplaza por la integral de una función de Rayleigh. La utilización de la mano de obra y las curvas acumuladas de utilización de la mano de obra se pueden calcular y luego extrapolar para incluir la mano de obra necesaria para detectar los errores restantes. No discutiremos este trabajo en detalle en este documento ya que utiliza muchos de los enfoques de confiabilidad aceptados y la herramienta aún está en desarrollo.

V. Conclusión Las

métricas para medir la confiabilidad del software existen y se pueden utilizar a partir de la fase de requisitos. En cada fase del ciclo de vida del desarrollo, las métricas pueden identificar áreas potenciales de problemas que pueden generar problemas o errores. Encontrar estas áreas en la fase en que se desarrollan reduce el costo y evita posibles efectos secundarios de los cambios, más adelante en el ciclo de vida del desarrollo. Las métricas utilizadas en forma temprana pueden ayudar en la detección y corrección de fallas en los requisitos que conducirán a la prevención de errores más adelante en el ciclo de vida. Se ha demostrado que los beneficios de costo de encontrar y corregir problemas en la fase de requisitos son al menos un factor de 14, lo que constituye un argumento sólido para seguir este enfoque y generar confiabilidad a partir de la fase de requisitos.

REFERENCIAS [1]

Musa, JD, A. Iannino y K. Okumoto, Software Reliability: Measurement, Prediction, Application, Professional Edition: Software Engineering Series, McGraw–Hill, Nueva York, NY, 1990.

[2] Este modelo fue desarrollado en conjunto con Sean Arthur de Virginia Tech y Darush Davani de la Universidad de Towson.

[2] Triantafyllos, G., S. Vassiliadis y W. Kobrosly, "Sobre la predicción de fallas de implementación informática a través de modelos de predicción de errores estáticos", Journal of Systems and Software, vol. 28, No. 2, febrero de 1995, págs. 129-142.

[3] Waterman, RE y LE Hyatt, “Testing - When Do I Stop?”, (Invitado) Conferencia Internacional de Pruebas y Evaluación, Washington, DC, octubre de 1994.

[4] Estándar IEEE 610.12-1990 Glosario de terminología de ingeniería de software

[5] Kitchenham, Barbara, Pfleeger, Shari Lawrence, Software Quality: The Elusive Target, IEEE Software 13, 1 (enero de 1996) 12-21.

[6] Gillies, AC, Software Quality, Theory and management, Chapman Hall Computing Series, Londres, Reino Unido, 1992.

[7] Estándar IEEE 982.2-1987 Guía para el uso del diccionario estándar de medidas para producir software confiable.

[8] Guía de garantía de software de la NASA, NASA GSFC MD, Oficina de seguridad y garantía de la misión, 1989.

[9] Departamento de Defensa. Documentación y desarrollo de software estándar militar (diciembre de 1994), MIL-STD-498.

[10] Wilson, W., Rosenberg, L., Hyatt, L., Análisis de calidad automatizado de las especificaciones de requisitos de lenguaje natural en Proc. Decimocuarta Conferencia Anual de Calidad de Software del Noroeste del Pacífico, Portland OR, 1996.

[11] Rosenberg, L. y Hammer, T., Métricas para el aseguramiento de la calidad y la evaluación de riesgos, Proc. Undécima Semana Internacional de la Calidad del Software, San Francisco, CA, 1998.

[12] Rosenberg, L., Métricas para entornos orientados a objetos, Tercera conferencia anual de métricas de software de EFAITP/AIE, 1997.

[13] Hammer, T., Rosenberg, L., Huffman, L., Hyatt, L., Pruebas de requisitos de medición en Proc. Conferencia internacional sobre ingeniería de software (Boston MA, mayo de 1997) IEEE Computer Society Press.

[14] Boehm, B. Tutorial: Gestión de riesgos de software (1989), IEEE Computer Society Press.