Revista Cubana de Ciencias Informáticas
Vol. 15, No. 4, Octubre-Diciembre, 2021
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 61-75

Artículo original

Estimación de la fiabilidad de software: Modelo Littlewood-Verall

Software reliability estimation: Littlewood-Verall Model

Geidis Sánchez Michel [1*] https://orcid.org/0000-0003-2488-558X
Serguey González Garay [1] https://orcid.org/0000-0001-5344-405X
Maykel Ramirez Reyes [1] https://orcid.org/0000-0002-1151-482X

[1] Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños km 2 ½Reparto Torrens. Boyeros. {gsanchez, sgaray, maykelrr}@uci.cu

*Autor para la correspondencia. (gsanchez@uci.cu)

RESUMEN

La tecnología ha crecido de forma sorprendente durante las últimas décadas y ha alcanzado gran poder la industria del software. El software abarca las más variadas ramas de la vida humana y de la industria en general; propiciando un elevado interés por la calidad. Uno de los aspectos más importante de la calidad del software es la fiabilidad; la cual puede ser medida o estimada mediante datos históricos. Para esta medición juegan un papel fundamental los modelos probabilísticos de fiabilidad. Los modelos, a diferencia de las normas, no contienen requisitos que deben cumplir los sistemas de gestión de la calidad sino directrices para la mejora. La aplicación de modelos aporta a las empresas o proyectos una visión general de su situación actual y su proyección futura. En este sentido, en la investigación se propone el modelo Littlewood-Verall para estimar el comportamiento de la fiabilidad del software en un proyecto. Se utilizaron como

métodos científicos el analítico sintético, la modelación y el método experimental. Los resultados obtenidos de la aplicación del método experimental evidenciaron una disminución del 55 % del número de fallos detectados en la etapa de despliegue en relación a la etapa de implementación a partir de la estimación del modelo propuesto.

Palabras clave: calidad; fallos; fiabilidad; modelo.

ABSTRACT

Technology has grown in an astonishing way during the last decades and the software industry has reached great power. Software encompasses the most varied branches of human life and industry in general, leading to a high interest in quality. One of the most important aspects of software quality is reliability, which can be measured or estimated using historical data. Probabilistic reliability models play a fundamental role in this measurement. Unlike standards, models do not contain requirements to be met by quality management systems, but rather guidelines for improvement. The application of models provides companies or projects with an overview of their current situation and their future projection. In this sense, the Littlewood-Verall model is proposed in the research to estimate the behavior of software reliability in a project. Synthetic analytical, modeling and experimental methods were used as scientific methods. The results obtained from the application of the experimental method showed a 55% decrease in the number of failures detected in the deployment stage in relation to the implementation stage based on the estimation of the proposed model.

Keywords: quality; failures; reliability; model.

Recibido: 08/07/2021

Aceptado: 16/08/2021

Revista Cubana de Ciencias Informáticas
Vol. 15, No. 4, Octubre-Diciembre, 2021
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 61-75

INTRODUCCIÓN

Los estudios de fiabilidad tradicionales han estado orientados al hardware. La mayor parte de los métodos y modelos desarrollados en esta disciplina se refieren básicamente a los elementos de hardware de los sistemas. Sin embargo, en los sistemas y equipos modernos el software desempeña un papel fundamental unido a la labor humana. Desde este punto de vista se podría estar hablando de tres tipos de fiabilidad: fiabilidad de hardware, fiabilidad humana y la fiabilidad del software (Solarte, y otros, 2009); siendo esta última la de interés en esta investigación.

La Norma ISO/IEC 25010 define la fiabilidad como la capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados. Esta característica se subdivide a su vez en las siguientes subcaracterísticas (ISO, 2021) :

1. **Madurez:** capacidad del sistema para satisfacer las necesidades de fiabilidad en condiciones normales.
2. **Disponibilidad:** capacidad del sistema o componente de estar operativo y accesible para su uso cuando se requiere.
3. **Tolerancia a fallos:** capacidad del sistema o componente para operar según lo previsto en presencia de fallos hardware o software.
4. **Capacidad de recuperación:** capacidad del producto software para recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallo.

La fiabilidad es la capacidad del software para cumplir los requisitos (Neufelder, 2018). Por su parte (Álvarez , 2019) expresa que la fiabilidad es la probabilidad de que el producto (dispositivos, sistemas y subsistemas) cumplan las funciones para las que fueron diseñadas, bajo

Editorial “Ediciones Futuro”
Universidad de las Ciencias Informáticas. La Habana, Cuba
rcci@uci.cu

especificaciones dadas.

Para (Lucero, y otros, 2020) el concepto más simple de fiabilidad, es aquel que comprueba que el producto cumple ciertas especificaciones, y cuando esto ocurre, es enviado al cliente.

(Gómez, y otros, 2020) en su libro “Aproximación a la Ingeniería del Software” hacen referencia a la fiabilidad como es el grado de ausencia de fallos durante la operación del producto software. Puede estimarse como el número de fallos producidos o el tiempo durante el que permanece inutilizable durante un intervalo de operación dado.

Otros estudios sostienen que la fiabilidad se refiere a la probabilidad de que el software funcione sin fallos durante un periodo de tiempo determinado en un entorno específico (Rowell, y otros, 2018) ; (Hanagal, y otros, 2021).

(Gradinaru, 2020) define la fiabilidad como la probabilidad de que el software se ejecute durante un periodo de tiempo determinado sin que se produzca un fallo, ponderada por el coste para el usuario de cada fallo que se produzca.

Como se muestra en la Tabla 1, son diversos los criterios que se utilizan por los autores para definir la fiabilidad de software, resaltando la importancia de que el software funcione libre de fallos.

Tabla 1 - Criterio de fiabilidad de software según autores.

Criterios	Referencias bibliográficas
Capacidad de un sistema para desempeñar las funciones especificadas.	(ISO, 2021); (Neufelder, 2018); (Lucero, y otros, 2020); (Castellor, 2017)
El grado de ausencia de fallos durante la operación del producto software.	(Gómez, y otros, 2020); (Lee, y otros, 2018)
La probabilidad de que el software funcione sin fallos.	(Alvarez , 2019); (Rowell, y otros, 2018); (Hanagal , 2019); (Gradinaru, 2020); (Rawat, y otros, 2017)

Los autores de la presente investigación asumen que, en las definiciones descritas sobre la fiabilidad del software, aparecen tres aspectos esenciales y que deben estar definidos de manera precisa:

1. ¿Qué función debe desempeñar?
2. ¿En qué condiciones debe desempeñar las funciones?
3. ¿Cuánto tiempo debe desempeñar de manera satisfactoria las funciones en esas condiciones?

Concluyen los autores que además de las definiciones anteriores, existen muchas otras, pero todas ellas relacionan un conjunto de características que determinan como elementos fundamentales de la fiabilidad del software la probabilidad de que el software funcione como se espera por el usuario, bajo determinadas condiciones en un periodo de tiempo determinado.

Modelos de estimación de la fiabilidad

Un modelo de crecimiento de la fiabilidad es un modelo numérico de la fiabilidad del software, que predice cómo debería mejorar la fiabilidad del software con el tiempo a medida que se descubren y reparan los errores (JavaTpoint, 2018).

El modelado de la fiabilidad del software ha ido madurando durante los últimos años hasta el punto de que se puede obtener información muy útil a la hora de desarrollar una aplicación informática, mediante la aplicación de un modelo determinado a cada problema. Son muchos los modelos que pueden ser empleados en el análisis de la fiabilidad de una aplicación informática, el principal problema radica en que no se puede encontrar un modelo que funcione bien en cualquier situación. Es tarea de los ingenieros adaptar un método o componer uno nuevo a partir de los ya existentes para conseguir una solución que se amolde al problema en particular que se está tratando (Lucero, y otros, 2020) .

Existen diversos modelos de crecimiento de fiabilidad que han sido derivados de experimentos de fiabilidad en varios dominios de aplicación, a continuación, se presentan una representación:

Modelo Jelinski-Moranda: fue uno de los primeros modelos de crecimiento de la fiabilidad del software. Asume que hay *n* defectos de software inicialmente al comienzo del procedimiento de prueba y cada uno de ellos es independiente de los demás defectos. Un defecto se elimina con seguridad y no se introducen otros nuevos defectos durante la fase de depuración (Kumar , y otros, 2017)**.**

Tiene como inconveniente que supone que las reparaciones del software se implementan siempre correctamente, sin embargo, en la práctica esto no sucede así ya que al reparar un defecto se pueden introducir uno o varios defectos.

Modelo Schick-Wolverton: se basa en la misma hipótesis que el modelo Jelinski-Moranda, salvo que se supone que la función de riesgo es proporcional al contenido de fallos actual del programa, así como al tiempo de fallo transcurrido desde el último fallo (Hanagal , 2019). Dado que se mantiene la hipótesis de que todos los defectos son similares y por tanto tienen la misma probabilidad de manifestarse, la intensidad de fallos del programa según este modelo simula que durante una primera fase es cuando se detectan un mayor número de defectos, alcanzándose un máximo a partir del cual disminuye el número de fallos que ocurren ya que se ha reducido el número de defectos existentes, como consecuencia de las acciones de corrección.

Este modelo tiene como inconveniente que consideran la independencia entre fallos, que el número de fallos que ocurren es igual al número de defectos que se corrigen, siendo por tanto un número finito, y que todos los defectos tienen igual probabilidad de manifestarse en un momento dado.

Modelo Littlewood-Verall: trata de tener cuenta la incertidumbre de la fiabilidad del sistema tras una acción de corrección de fallos. La idea subyacente es que cada acción de reparación puede llevar al sistema a un estado mejor o peor en comparación con el estado anterior del sistema en

funcionamiento (Stefano, y otros, 2018) . Este modelo tiene en cuenta los inconvenientes del modelo propuesto por Jelinski-Moranda introduciendo un elemento aleatorio en la mejora del crecimiento de la fiabilidad conseguida por una reparación del software. Además, modela el hecho que a medida que los defectos son reparados, el promedio de mejora en cuanto a fiabilidad por reparación disminuye.

A continuación, en la Tabla 2 se realiza una comparación de los modelos antes mencionados teniendo en cuenta los criterios de crecimiento de la fiabilidad, corrección de errores y probabilidad de los defectos a partir del comportamiento probabilístico de los diferentes modelos.

Tabla 2 - Tabla comparativa de los modelos de estimación de la fiabilidad.

Modelo	Crecimiento de la fiabilidad	Corrección de errores	Probabilidad de los defectos
Jelinski-Moranda	constante	perfecta	igualmente probables
Schick-Wolverton	constante	perfecta	igualmente probables
Littlewood-Verall	creciente/decreciente	imperfecta	diferente probabilidad

Del análisis de la tabla 2 se puede llegar a la conclusión que los modelos Jelinski-Moranda y Schick-Wolverton consideran que los fallos de un sistema son independientes entre sí, que la intensidad del proceso de fallo es proporcional al número de defectos existentes en el sistema, y que la corrección es perfecta. Cada defecto tiene la misma probabilidad de manifestarse, siendo constante entre fallos, y por tanto también es constante entre fallos la intensidad de fallos del sistema, la cual disminuye cada vez que se produce un fallo en una cantidad constante, ya que se supone que al producirse un fallo siempre se corrige el defecto que lo origina, y además no se

Revista Cubana de Ciencias Informáticas
Vol. 15, No. 4, Octubre-Diciembre, 2021
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 61-75

cometen nuevos errores. Siendo en el proceso de desarrollo de software poco probable que los sistemas mantengan ese comportamiento. Por lo que se propone emplear el modelo Littlewood-Verall el cual establece que la intensidad de los fallos no decrece de manera constante cada vez que se produce un fallo, sino que considera que la intensidad de fallos es también una variable aleatoria, que depende del número de fallos.

MÉTODOS O METODOLOGÍA COMPUTACIONAL

En esta investigación se emplearon como métodos científicos el analítico sintético, la modelación y el método experimental. El método analítico sintético permitió el análisis y síntesis de la literatura relacionada con la fiabilidad de software y los modelos de estimación de la fiabilidad. La modelación permitió representar la aplicación del modelo de estimación de la fiabilidad. El método experimental se utilizó para desarrollar dos experimentos, uno de tipo preexperimento estudio de casos con una sola medición y el otro de Pre y Post prueba con un solo grupo.
Teniendo en cuenta el análisis de los modelos se decide utilizar en la propuesta de solución para estimar la fiabilidad del software el modelo **Littlewood-Verall**.

Este modelo intenta tener en cuenta la generación de defectos en el proceso de corrección considerando la posibilidad de que un programa sea menos fiable al corregir un defecto. El mismo introduce una diferencia importante respecto a los modelos descritos anteriormente, los cuales asumen un proceso de depuración de defectos perfecto.

Formalmente, las premisas de este modelo son:

1- Tiempo de ejecución sucesivo entre fallas; por ejemplo: X_i, i=1, 2, 3..., son variables aleatorias independientes con funciones de densidad probable.

$$f(X_i)=\lambda_i e^{-\lambda_i X_i}$$

Editorial "Ediciones Futuro"
Universidad de las Ciencias Informáticas. La Habana, Cuba
rcci@uci.cu

Revista Cubana de Ciencias Informáticas
Vol. 15, No. 4, Octubre-Diciembre, 2021
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 61-75

Donde λ_i es la tasa de fallo. Se asume que X_i es exponencial con λ_i como parámetro.

2- λ_i forma una secuencia de variables aleatorias, cada una con una distribución gamma de parámetros αy $\Psi(i)$, como sigue:

$$g(\lambda_i) = \frac{[\Psi_i]^{\alpha}\lambda_{i^{\alpha-1}}e^{-\Psi(i)\lambda_i}}{\Gamma(\alpha)}$$

$\Psi(i)$ es una función creciente del número de defectos i, que describe la "calidad" del programador y la dificultad de la tarea de programación. Un buen programador debe tener una función creciente $\Psi(i)$ más rápida que un programador de menos nivel. Al requerir que $\Psi(i)$sea creciente, la condición

$$P(\lambda(i)<X)>P(\lambda(i-1)<X)$$

se satisface para toda i. Esto refleja que la intención es construir un programa mejor después de que un defecto sea detectado y corregido. También refleja la realidad de que en ocasiones las correcciones harán que el programa sea peor. Para la función $\Psi(i)$, Littlewood and Verrall sugieren que cualquiera de las dos formas: β_0+$\beta_0 i$ ó β_0+$\beta_0 i^2$. Asumiendo a priori una distribución uniforme para α, los parámetros β_0 y β_1 pueden ser encontrados por la estimación de máxima verosimilitud.

3- Durante la prueba, el software se ejecuta bajo condiciones similares a como se prevee que funcione.

El tiempo medio entre la falla $(i-1)$ y la $i$ésima falla,$\Theta(i)$, se expresa:

$$\Theta(i) = \frac{t_i+\Psi(i)}{\alpha}$$

α es un parámetro de la distribución gamma para la intensidad de fallas.

RESULTADOS Y DISCUSIÓN

Editorial "Ediciones Futuro"
Universidad de las Ciencias Informáticas. La Habana, Cuba
rcci@uci.cu

Se realizaron dos experimentos en el proyecto teniendo en cuenta las etapas de implementación y despliegue con el objetivo de analizar el futuro comportamiento del sistema ante posibles fallos. Tabla 3

Etapa de Implementación

Experimento 1: Análisis de la tolerancia ante fallos en la etapa de pruebas.

G: Grupo de experimentación compuesto por el proyecto.

X: Análisis de la tolerancia ante fallos en la etapa de pruebas.

O1: Observación del total de fallos detectados en la etapa de pruebas.

G X O1

Revista Cubana de Ciencias Informáticas
Vol. 15, No. 4, Octubre-Diciembre, 2021
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 61-75

Tabla 3- Fallos detectados en las pruebas funcionales.

No. error	Tiempo entre fallos	No. error	Tiempo entre fallos	No. error	Tiempo entre fallos	No. error	Tiempo entre fallos	No. error	Tiempo entre fallos
1	3	17	3	33	2	49	21	65	31
2	2	18	1	34	6	50	14	66	24
3	4	19	2	35	5	51	24	67	21
4	3	20	2	36	1	52	30	68	13
5	4	21	4	37	2	53	5	69	34
6	1	22	3	38	2	54	27	70	28
7	3	23	4	39	3	55	19	71	35
8	2	24	1	40	1	56	7	72	15
9	3	25	2	41	5	57	13	73	25
10	1	26	3	42	6	58	20	74	37
11	5	27	2	43	6	59	4		
12	3	28	2	44	10	60	13		
13	2	29	3	45	9	61	18		
14	4	30	5	46	13	62	27		
15	4	31	1	47	15	63	9		
16	2	32	4	48	7	64	15		

Utilizando el juego de datos de los fallos detectados en las pruebas funcionales se aplicó el modelo de Littlewood-Verall. Utilizando la herramienta SMERFS3 se obtuvieron los siguientes resultados representados en la Figura 1, en el eje de las abscisas se manifiestan los fallos detectados y en el eje de las ordenadas el tiempo entre fallos.

Editorial "Ediciones Futuro"
Universidad de las Ciencias Informáticas. La Habana, Cuba
rcci@uci.cu

Revista Cubana de Ciencias Informáticas
Vol. 15, No. 4, Octubre-Diciembre, 2021
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 61-75

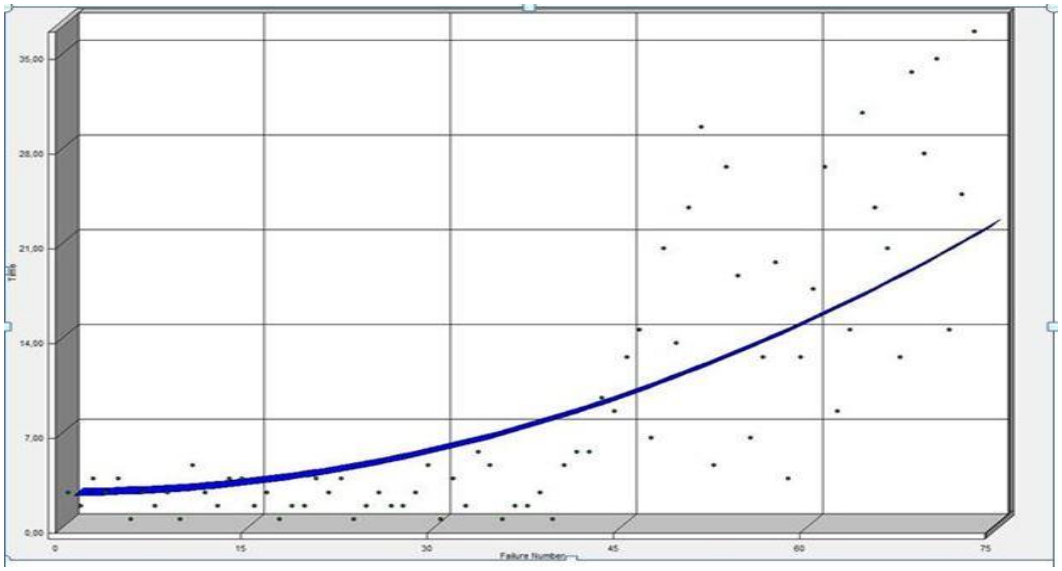

Fig. 1 - Resultados del método Littlewood-Verall en etapa implementación.

En la Figura 1 mediante una gráfica se obtiene una función creciente como resultado del método Littlewood-Verall, donde se evidencia en su comportamiento que a medida que aumentan los fallos detectados a la vez aumenta el tiempo entre fallos permitiendo de esta manera reflejar el posible comportamiento antes posibles fallos que puedan ocurrir.

Etapa de despliegue

Experimento 2. Pre y Post prueba con un solo grupo

G: Grupo de experimentación compuesto por el proyecto.

X: Análisis de la tolerancia ante fallos.

O1: Observación del total de fallos detectados en la etapa de prueba.

O2: Observación del total de fallos detectados en la etapa de despliegue.

G O1 X O2

A continuación, en la Tabla 4 se muestran los fallos detectados en la etapa de despliegue.

Editorial "Ediciones Futuro"
Universidad de las Ciencias Informáticas. La Habana, Cuba
rcci@uci.cu

Revista Cubana de Ciencias Informáticas
Vol. 15, No. 4, Octubre-Diciembre, 2021
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 61-75

Tabla 4. Fallos detectados en la etapa de despliegue.

No. error	Tiempo entre fallos	No. error	Tiempo entre fallos	No. error	Tiempo entre fallos
1	60	17	600	33	6033
2	90	18	680		
3	105	19	700		
4	200	20	777		
5	185	21	810		
6	305	22	970		
7	375	23	900		
8	400	24	1000		
9	413	25	1020		
10	499	26	2000		
11	500	27	3025		
12	535	28	4087		
13	230	29	4098		
14	180	30	5010		
15	375	31	6000		
16	400	32	6033		

Utilizando el juego de datos de los fallos detectados en la etapa de despliegue se aplicó el modelo de Littlewood-Verall. Utilizando la herramienta SMERFS3 se obtuvieron los siguientes resultados representados en la Figura 2, en el eje de las abscisas se manifiestan los fallos detectados y en el eje de las ordenadas el tiempo entre fallos.

Editorial "Ediciones Futuro"
Universidad de las Ciencias Informáticas. La Habana, Cuba
rcci@uci.cu

Revista Cubana de Ciencias Informáticas
Vol. 15, No. 4, Octubre-Diciembre, 2021
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 61-75

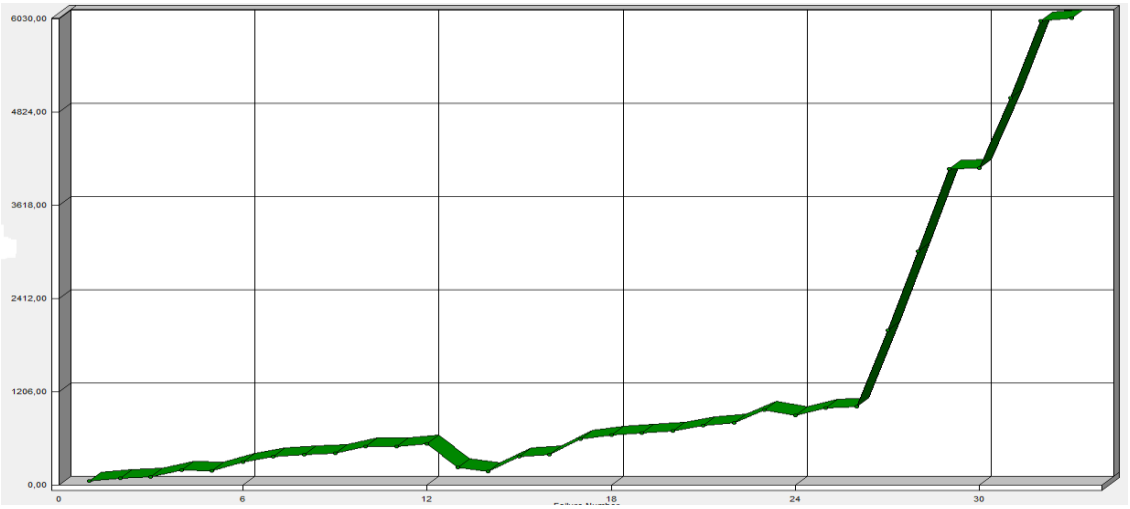

Fig. 2- Resultados del método Littlewood-Verall en etapa de despliegue.

Se puedo observar que los resultados obtenidos en la etapa de despliegue son superiores a los obtenidos en la etapa de implementación cuando se analizan los fallos detectados en ambas etapas, disminuyendo el número de fallos en 41 lo que representa un 55% de los fallos detectados en la etapa de implementación. Se evidencia mediante la gráfica una función con un crecimiento más lento de los fallos detectados con respecto al tiempo entre fallos, correspondiéndose este comportamiento con la observación realizada anteriormente respecto a ambas etapas de implementación y despliegue y así lo demuestran ambas funciones mediante las gráficas.

CONCLUSIONES

Al definir la fiabilidad como una probabilidad lleva implícito que los resultados solo puedan obtenerse de tratamientos estadísticos basados en pruebas y la estimación de dichos resultados tendrá el riesgo asociado al tratamiento estadístico. Por tal motivo, resulta especialmente interesante la cuantificación de dicha fiabilidad, de forma que sea posible hacer estimaciones sobre el comportamiento del software.

La aplicación del modelo Littlewood-Verall en un proyecto real demostró la necesidad en las pruebas de software de recopilar datos precisos del comportamiento del sistema para estimar su comportamiento en la etapa de despliegue. Evidenciando a través de la modelación del comportamiento del software un salto de fiabilidad en la etapa de despliegue comparado con la etapa de implementación. Lo que permitió considerar un mejor desempeño del software en su vida útil.

REFERENCIAS

Álvarez . fiabilidad, Estado del arte de la ingeniería en. 2019. 323, s.l. : Técnica Industrial. La revista de la ingeniería de la rama industrial, 2019, págs. 50-58.

Castellor, . 2017. UNED - Sistemas en tiempo real. UNED - Sistemas en tiempo real. [Online] 2017. https://uned-sistemas-tiempo-real.readthedocs.io/es/latest/tema02.html.

Gómez, Sebastián Rubén and Moraleda Gil, . 2020. Aproximación a la Ingeniería del Software. Segunda . s.l. : Centro de Estudios Ramon Areces SA, 2020.

Gradinaru, . 2020. Linkedin. Linkedin. [Online] 18 Septiembre 2020. [Cited: 19 Mayo 2021.] https://www.linkedin.com/pulse/software-reliability-principles-practices-glenford-myers-gradinaru?trk=read_related_article-card_title.

Hanagal . model, Modeling on generalized extended inverse Weibull software reliability growth. 2019. s.l. : Journal of Data Science, 2019, Vol. 17. 3.

Hanagal, David D. and Bhalerao, Nileema N. . 2021. Software Reliability Growth Models. [ed.] Springer Nature. Singapore : s.n., 2021.

ISO. 2021. ISO 25000. ISO 25000. [Online] ISO, 2021. [Cited: 19 Mayo 2021.] https://iso25000.com/index.php/normas-iso-25000/iso-25010?start=3.

JavaTpoint. 2018. JavaTpoint. JavaTpoint. [Online] 2018. [Cited: 20 Mayo 2021.] https://www.javatpoint.com/software-engineering-software-reliability-models.

Kumar , y otros. Systems, Analysis of Various Software Reliability Models and Proposing a New Model of Software Reliability for Embedded. 2017. 3, s.l. : International Journal of Innovative Research in Computer Science & Technology, 2017, Vol. 5.

Lee, y otros. A Software Reliability Model Considering the SyntaxError in Uncertainty Environment, Optimal ReleaseTime, and Sensitivity Analysis. 2018. 9, s.l. : Applied Sciences, 2018, Vol. 8.

Lucero, y otros. Fiabilidad en la Calidad del Software: Modelos, Métodos y Estrategias. 2020. San Luis : XXII Workshop de Investigadores en Ciencias de la Computación, 2020.

Modelos de Calidad para procesos de software. Solarte, , Muñoz, and Arias, . 2009. 42, Pereira : s.n., 2009, Scientia Et Technica,, Vol. XV, pp. 375-379.

Neufelder, Ann Marie . 2018. Ensuring Software Reliability. [ed.] CRC Press. New York : Marcel Dekker, 2018.

Rawat, y otros. development, Mangey Software reliability growth modeling for agile software. 2017. 4, 2017, Vol. 27, pp. 777–783.

Rowell, and Kolsti, . 2018. Software Reliability Fundamentals forInformation Technology Systems. s.l. : Wright-Patterson AFB, 2018. 39.

Stefano, y otros. centers, Characterizing machines lifecycle in Google data. 2018. s.l. : Performance Evaluation, 2018, Vol. 126.

Conflicto de interés

Los autores autorizan la distribución y uso de su artículo.

Contribuciones de los autores

Conceptualización: Geidis Sánchez Michel

Curación de datos: Geidis Sánchez Michel

Análisis formal: Serguey González Garay

Adquisición de fondos: -

Investigación: Geidis Sánchez Michel

Metodología: Serguey González Garay

Administración del proyecto: Maykel Ramirez Reyes

Recursos: Serguey González Garay

Software: --

Supervisión: Maykel Ramirez Reyes

Validación: Geidis Sánchez Michel

Visualización: Maykel Ramirez Reyes

Redacción – borrador original: Geidis Sánchez Michel

Redacción – revisión y edición: Geidis Sánchez Michel