Conceptos básicos del aseguramiento de la calidad del *software*: contrastes entre un enfoque tradicional y otro moderno

David Jesús Pérez Camacho y Jenny Rodríguez Vargas

Escuela de Ingeniería,
Universidad Latinoamericana de Ciencia y Tecnología,
ULACIT, Urbanización Tournón, 10235-1000
San José, Costa Rica
dperezc142, jrodriguezv419@ulacit.ac.cr
http://www.ulacit.ac.cr

Resumen Desde los años setenta han existido metodologías de desarrollo de *software* que buscan dirigir el ciclo de vida de una idea hasta un producto final. Estos métodos buscan garantizar que el proceso y el resultado sean conformen a especificaciones acordadas entre el cliente y el equipo de trabajo. Las metodologías tradicionales de desarrollo aseguran la calidad hacia el final del ciclo de vida, mientras que una nueva corriente de desarrollo, denominada ágil, desafía el paradigma tradicional y busca identificar y corregir errores lo más pronto posible. En comparación, estudios estadísticos demuestran que las metodologías ágiles reducen los problemas de calidad y son más eficientes a la hora de desarrollar sistemas informáticos.

Keywords: ágil, cascada, aseguramiento de la calidad del *software*, pruebas de *software*, metodologías de desarrollo.

1. Introducción

El aseguramiento de la calidad, dentro de sus muchos significados, puede definirse como la conformidad de un producto con las especificaciones y el diseño, e incluye todas las etapas del ciclo de vida (Edrandaly, 2008). Esto permite incrementar las posibilidades de que el proceso de desarrollo de un producto o servicio satisfaga las necesidades de los usuarios, con base en estándares e indicadores para dar seguimiento a las variaciones de la calidad.

Es importante entender que el aseguramiento de la calidad de los productos o servicios puede ser un factor determinante del éxito o fracaso comercial de una empresa. La percepción de los usuarios finales resulta de gran valor en la evaluación de la calidad de los productos (Ahamed, 2011), sobre todo en esta época en que los consumidores pueden expresar sus apreciaciones y críticas en las redes sociales y foros de opinión.

Como consecuencia, las empresas dedicadas al desarrollo de *software* deben contar con metodologías y herramientas que les permitan medir y asegurar la calidad de los sistemas que producen. Esto resulta de utilidad tanto para mantener a sus clientes actuales como para atraer nuevos, en un mercado competitivo como el actual, lo cual a largo plazo garantizará su estabilidad y crecimiento comercial. Entonces, las metodologías de aseguramiento de la calidad garantizan que las etapas del ciclo de vida del desarrollo de los sistemas se

ejecutan siguiendo procedimientos rigurosos, y que el producto final cumple con los requerimientos establecidos por los usuarios.

Por ejemplo, la metodología en cascada se considera la forma tradicional de abordar el desarrollo de *software*. A partir de ella surgieron otras que intentaron mejorar su enfoque, orientado a una estructura fija, basada en hitos y lineal.

Por otro lado, a partir del 2001, surge un movimiento que se denomina Ágil y cambia la manera de desarrollar *software* y aborda el ciclo de vida del producto y su calidad desde un punto de vista distinto.

Los beneficios de utilizar procesos de aseguramiento de la calidad correctamente definidos incluyen la satisfacción del cliente, mejora o conservación de la reputación en el mercado, cumplimiento de las regulaciones legales y estándares tecnológicos, y el crecimiento económico de las empresas. En tanto, la falta de calidad de los sistemas informáticos genera críticas, devoluciones de productos, gastos operativos, pérdida de imagen y hasta posibles demandas.

Las consecuencias de no asegurar la calidad del *software* desde la etapa inicial pueden incluir, entre otras, la cancelación del proyecto, atrasos en el calendario y costos excesivos (Casper, J. 2015). Se considera que estos resultados no son deseables para ninguna empresa dedicada al desarrollo de aplicaciones informáticas, principalmente porque tienen un impacto económico negativo y generan una mala imagen.

El objetivo de este trabajo es evaluar las metodologías que se usan para la gestión del aseguramiento de la calidad de *software*, comparando una metodología tradicional con otra ágil. Este artículo de investigación pretende identificar y exponer esas metodologías y sus conceptos básicos que han logrado que los sistemas informáticos se conformen de manera exitosa con sus respectivas especificaciones, tomando en cuenta tanto los procesos como el producto final (ciclo de vida de un proyecto).

Esto se logra mediante una revisión bibliográfica, utilizando artículos de EBSCO y Google Scholar referentes al aseguramiento de la calidad de *software*.

Al identificar, entender y poner en práctica los conceptos, las habilidades y los recursos de la disciplina del aseguramiento de la calidad, los gerentes y asociados de cualquier empresa pueden aumentar su probabilidad de éxito. Los sistemas que se desarrollen permitirán un mejor mantenimiento de la aplicación, cumplirán con estándares informáticos y satisfarán las necesidades de los usuarios finales.

2. Desarrollo

Para decidir cuáles metodologías comparar se utilizaron fuentes bibliográficas en los portales de EBSCO y Google Scholar que estuvieran relacionadas con el

aseguramiento de la calidad en el desarrollo de *software*. El enfoque principal es contrastar el enfoque tradicional versus el enfoque moderno o "ágil".

Uno de los criterios utilizados en la selección de las fuentes fue la antigüedad (no mayores a diez años), esto con el fin de tener un panorama moderno. Cuando fue posible, se escogieron fuentes completas de revistas internacionales, con el fin de tener visiones cosmopolitas.

Cabe mencionar que la búsqueda de fuentes relacionadas con el tema de investigación son limitadas. Como este trabajo pretende dar un enfoque general, se buscaron aproximadamente 25 fuentes de las que se seleccionaron 14 por su similitud en las palabras claves, entre ellas, "desarrollo orientado a pruebas", "metodologías de desarrollo de *software*", "ciclo de vida del *software*", "casos de uso", "metodologías agiles", entre otras.

Al hacer esto, se identificó que la metodología tradicional de desarrollo es la denominada cascada, la cual fue mejorada con herramientas como el modelo entidad-relación, los diagramas de flujo de datos, entre otros (Rivadeneira, 2012). Asimismo, aparece en el siglo XXI una corriente de desarrollo que buscaba hacer algo completamente distinto, lo que dio nacimiento al desarrollo Ágil (Duncan & Peercy, 2012).

A continuación se presenta un breve desarrollo de estos temas.

2.1. Ciclo de vida tradicional del *software*

El ciclo de vida de desarrollo del *software* provee una estructura para que tanto los usuarios como el equipo de operaciones entiendan el progreso del proyecto. Permite que todos los interesados tengan visibilidad del estado actual y puedan tomar decisiones que los lleven a alcanzar los objetivos del negocio (Jain & Jain, 2011).

Uno de los propósitos de establecer metodologías de ciclo de vida del *software* es aumentar la productividad y calidad de los sistemas desarrollados. En ocasiones, se aborda el ciclo de vida desde la perspectiva de la administración de proyectos, que ayuda a la entrega en tiempo, alcance y costos apropiados (Jain & Jain, 2011). Uno de los métodos tradicionales del ciclo de vida del *software* es el llamado Cascada, que se atribuye a Walter Royce y data de 1970 (Petersen et al., 2009). Tiene una estructura definida en hitos, y una etapa debe cumplirse para que continúe la siguiente. De manera general, las etapas son:

- Requerimientos: se documentan los requerimientos del cliente y se crea un diseño de alto nivel.
- Diseño e implementación: se crea y documenta la arquitectura del sistema.
- Pruebas: se prueba el sistema en diferentes ambientes y se busca la conformidad con los requerimientos del cliente.
- Lanzamiento: se lleva al sistema a un punto en el que se puede distribuir junto a la documentación respectiva.
- Mantenimiento: el cliente utiliza el sistema y según sea necesario, en especial por errores en ejecución, se le da mantenimiento.

Según Jain & Jain, y esto concuerda con Petersen et al., el mayor esfuerzo relacionado con la calidad se realiza luego de que los requerimientos se recopilan

y se implementa la solución. Aunque este escenario puede ser el más común, es importante notar que existen metodologías orientadas a las pruebas, en las que estas se definen aun antes de escribir el código.

Aunque las etapas de este ciclo de vida poseen una estructura completa, una de las críticas a este modelo es que genera una alta incidencia de errores de compilación por hacer las pruebas una vez finalizada la etapa de desarrollo, o sea, se efectúan independientemente. Esto atrasa la resolución de problemas y el lanzamiento del sistema. Las metodologías ágiles han surgido como una alternativa a las fallas en el modelo de cascada (Petersen et al., 2009).

2.2. Desarrollo orientado a pruebas

Esta es una técnica en la que las pruebas y los casos de uso se especifican antes de iniciar el ciclo de desarrollo; o en otras palabras, se establecen las pruebas antes de que se escriba el código. Existen diversas opiniones en cuanto a su propósito, entre ellas que ayuda a establecer requerimientos claros entre el usuario y el desarrollador, otra es que permite escribir código limpio que funciona. Sus partidarios dicen que lleva a una menor cantidad de defectos, menos depuraciones y un mejor diseño. Sin embargo, los escépticos de esta técnica argumentan que es difícil de aprender y que el enfoque es basado en casos de uso, aunque útil para identificar errores al nivel de código, puede dejar de lado problemas a nivel de diseño (Dogsa, 2011).

TDD (*Test Driven Development*), se compone de unos pasos básicos e iterativos, que se citan a continuación:

1. El desarrollador escribe un caso de prueba que define una nueva funcionalidad por implementar.
2. El desarrollador genera el código necesario para pasar la prueba.
3. El desarrollador corre la prueba. Si falla, se redefine el código y sigue a la siguiente funcionalidad. De lo contrario, se redefine el código hasta pasar la prueba.
4. El desarrollador periódicamente corre todos los casos de prueba para asegurarse de que el nuevo código no causa problemas con el anterior. Esto se llama pruebas de regresión.

Según Ficco et al., se han analizado varios estudios que, en su mayoría, exponen resultados positivos relacionados con código de mayor calidad y disminución de la densidad de defectos, aunque a su vez hay otros estudios que muestran resultados planos en temas de productividad cuando los programadores involucrados tenían poca experiencia con TDD. Por su parte, Dogsa et al. realizaron un estudio con proyectos a nivel industrial aplicando TDD y concluyen que esta técnica permite una regresión que poco afecta y lleva a un mejor diseño, dado que el programador escribe tan solo el código necesario para que funcione el *software*. Esto lleva a una fase de mantenimiento mucho más corta. Una limitación que ellos mismos hacen es que se debe tomar en cuenta que su estudio fue específico al área industrial.

2.3. Desarrollo ágil y el aseguramiento de la calidad

El desarrollo Ágil fue un movimiento que inicio a principios del siglo XXI como respuesta, en parte, a la cantidad de errores que generaban los métodos estándar de desarrollo de *software* (entre ellos, cascada). Ágil promueve la prevención de errores en vez de su corrección.

Esta metodología tiene su propio manifiesto, que cuenta con cuatro valores:

1. Individuos e interacciones por encima de procesos y herramientas.
2. *Software* que funciona por encima de documentación completa.
3. Colaboración con el cliente por encima de negociaciones contractuales.
4. Responder al cambio por encima de seguir un plan.

La idea central de Ágil es adaptarse al cambio y mantener una flexibilidad saludable durante el proceso de desarrollo, que no debe confundirse con un abandono total de estándares y practicas estructuradas. Se enfoca en el cliente y sus necesidades, teniendo un énfasis fuerte en la comunicación constante. Además, busca el desarrollo de pequeños hitos con mayor frecuencia por medio del uso de iteraciones y lanzamientos cortos, comunicados al cliente y probados desde el inicio, lo cual busca disminuir la incidencia de errores.

En su reporte "Manifiesto del Caos", Standish Group determinó, por medio de un estudio estadístico, que los trabajos de programación efectuados con métodos agiles fueron más probables de finalizar bien que los realizados con las metodologías tradicionales, como es el caso de cascada. También descubrió que los objetivos definidos en los proyectos que usaron la metodología ágil resultaron mejor planificados que los de cascada (Fiallos, 2015).

Una diferencia muy importante entre estas dos metodologías es que en Ágil las tareas se van ejecutando por medio de iteraciones, mientras por otro lado cascada define las tareas a largo plazo, lo cual afecta el costo, el tiempo, los recursos y la calidad.

Según Rivadeneira algunas comparaciones entre ambas metodologías son:

- En Ágil se toman en cuenta las necesidades de los clientes ya que se está en constante comunicación con ellos y se incluye como parte del equipo de desarrollo. En Cascada el involucramiento del cliente se da por medio de reuniones al inicio del proyecto, pero no se incluye durante el desarrollo y las pruebas.
- En Ágil durante la entrega de cada iteración se van realizando cambios para satisfacer al cliente y las iteraciones son cortas. Mientras que en Cascada los cambios son menos flexibles por seguir una estructura lineal y se debe terminar una etapa para iniciar la siguiente.
- Ágil se enfoca en finalizar cada tarea del proyecto eficazmente y no con mucha documentación. En Cascada se documenta demasiado.

De acuerdo con Duncan & Peercy y Jamieson & Fallah, la metodología Ágil asegura la calidad del *software* previniendo errores, en vez de corregirlos, al ejecutar las siguientes prácticas:

1. Colaboración: Durante la etapa de desarrollo se pueden realizar los cambios necesarios para ir efectuando lo señalado en la definición del proyecto, y de esta manera se evitan futuros problemas, los cuales se logran determinar por medio de los avances facilitados en cada fecha definida y así se va comprobando el progreso del cronograma para cumplir con el tiempo de entrega establecido.
2. Compromiso: Los involucrados se comprometen a alcanzar objetivos específicos, realistas a las capacidades de cada miembro del equipo y visible en todo momento. Lograr metas, así genera confianza entre todos.
3. Comunicación: Ayuda a los programadores a conocer las necesidades de los clientes (esto se adquiere por medio de una continua comunicación entre ambas partes).
4. Planeamiento: Al planificar las tareas que se deben llevar a cabo se previenen imprevistos los cuales pueden perjudicar la entrega del sistema.
5. Desarrollo: Realizar las tareas de programación de manera iterativa (es decir, se divide por etapas que se van repitiendo hasta completar la última actividad planificada), contribuye a mejorar la eficacia y la eficiencia de los sistemas para incrementar su calidad.
6. Entrega: Se realiza la entrega de un producto por cada iteración. Se establece una "definición de terminado", contra la que se puede evaluar el resultado obtenido en ese hito.
7. Reflexión: El equipo de trabajo aprende de los errores encontrados al utilizar la práctica de la retrospectiva, en la que se reúnen y analizan lo que funcionó, lo que se pudo mejorar y lo que no se ha intentado.

El aseguramiento de la calidad de *software* se debe implementar en todas las etapas del ciclo de vida del desarrollo de los proyectos para lograr cumplir con los requisitos definidos y así asegurar que los trabajos están elaborados con eficacia para satisfacer las necesidades de los clientes, los cuales pueden quedar satisfechos con las entregas realizadas.

Además se cerciorar de que se cumplan las políticas e instrucciones de cualquier organización para ejecutar los procesos establecidos y así mejorar la calidad y se deben efectuar estudios de los planes del desarrollo y de las pruebas por ejecutar y los resultados de esas revisiones se deben documentar (Owens, D.& Khazanchi, D., 2009).

Es importante que un gerente de calidad logre ordenar a los miembros de un equipo de trabajo de acuerdo con las diferentes competencias obtenidas, para que estos laboren organizadamente en las distintas fases de desarrollo de los programas computacionales y se pueda asegurar satisfacción, efectividad y eficacia en la finalización del *software*, esto debido a que un gerente se encarga de asignar las diferentes responsabilidades y tareas dentro del grupo de trabajo, las cuales deben conseguir un mismo objetivo y simplicidad en la labor efectuada para obtener resultados apegados al diseño establecido y al mismo tiempo se puede comprobar su respectivo progreso; sea este satisfactorio o no y tomar las mejores decisiones para solucionar lo que esté fallando.

En ocasiones puede resultar difícil desarrollar *software* y utilizar principios del aseguramiento de la calidad. Si bien existe literatura que indica qué hacer,

no siempre queda claro cómo hacerlo. De cierta manera Ágil ayuda a dar claridad en este aspecto al aplicar prácticas sencillas que se pueden usar durante el ciclo de vida del *software*, por ejemplo, iteraciones breves e incrementales (García et al., 2010), constante comunicación entre miembros del equipo y el cliente, entre otras.

Según Owens, D., & Khazanchi, D., algunas de las funciones del aseguramiento de la calidad son:

- Planear proyectos de *software* para determinar las diferentes actividades por realizar.
- Verificar las necesidades de los usuarios para poder entregarles un producto satisfactorio.
- Se deben establecer modelos de codificación para capturar los datos eficientemente.
- Se deben poseer las pruebas integradas para corroborar que el *software* se está ejecutando e implementando satisfactoriamente.

La clave de las metodologías agiles en el desarrollo de *software* es suscitar el trabajo en equipo y el aprendizaje de los desarrolladores para que se produzca un ambiente laboral agradable. También ayudan en la sencillez de los resultados obtenidos ya que debido a las constantes entregas de los avances se pueden ir realizando las respectivas mejoras, las cuales contribuyen a facilitar los trabajos realizados brindando procesos eficaces (los cuales se pueden descomponer en tareas pequeñas) para satisfacer las necesidades de los clientes y los objetivos de las empresas (Jamieson & Fallah, 2012).

Las metodologías ágiles permiten planificar y darles seguimiento a los proyectos de una forma más ordenada o controlada por medio de los requisitos establecidos y así se verifica si se efectúa el plan de las tareas definidas y se comprueban los resultados logrados para validar si se cumplió o no con los objetivos y garantizar, al momento de hacer la entrega final, si se cumple con alta calidad o no.

Los métodos ágiles resultan mejores en temas de calidad que el modelo estándar de programación porque se enfocan en reducir los costos de operación y el tiempo de implementación al planificar el ciclo de vida de tal manera que los atrasos en la fecha de entrega del producto sean mínimos. Además, el modelo estándar resulta rígido y por lo general es poco flexible con los cambios en medio del proyecto, esto porque no se produce la constante comunicación con el usuario para determinar sus necesidades de la manera que se efectúa en las metodologías ágiles.

3. Conclusión

Las metodologías de programación se crearon para dar secuencia lógica al desarrollo de *software*, así como para asegurar la calidad, es decir, lograr que tanto los procesos como el producto final estén conforme a lo requerido por los usuarios e interesados.

Para desarrollar una solución de *software* se necesita una metodología de ciclo de vida. Desde la década de los setenta empezaron a surgir metodologías y estructuras que guiaban a los equipos de desarrollo a alcanzar metas que los llevaran al producto final y que les permitiera hacerlo libre de errores.

Desde los primeros años de la programación hasta hoy, el modelo cascada se ha convertido en el tradicional en los proyectos de desarrollo. Esta metodología se basa en hitos e iteraciones completas, en las que la siguiente depende directamente de la anterior. En lo que concierne a la calidad, la metodología en cascada revelaba problemas de forma tardía y acumulada. Esto causaba retrasos y costos adicionales en los proyectos.

El denominado desarrollo Ágil, por el contrario, fue el resultado de querer hacer las cosas diferente. El ciclo de vida Ágil se basa en iteraciones incrementales y breves, las cuales incluyen pruebas y entregables en cada hito. Al construir poco a poco, y probar a medida que se construye, los errores se pueden encontrar y corregir desde el inicio; en realidad, se pueden identificar y solucionar en cualquier etapa del desarrollo.

Adicionalmente, las metodologías Ágiles se enfocan en prácticas que fomentan una fuerte comunicacion, colaboración, mejora continua, transparencia, inclusión del cliente de principio a fin y entregas constantes. Estadísticamente, según las fuentes consultadas, el uso de metodologías Ágiles genera una mejor calidad (menos errores en el producto de *software*), en comparación a metodologías tradicionales como cascada.

Sin embargo, aunque las fuentes señalan que las metodologías Ágiles generan menos problemas de calidad, se considera que hubiera sido provechoso tener acceso al informe del Standish Group para poder evaluar en qué entornos el desarrollo Ágil tiene tanto éxito y el de cascada resulta tan ineficiente. Los autores solo tuvieron acceso a un documento que hace referencia a tal informe, pero los resultados reales no son de libre distribución. Además, se considera que tener experiencia en ambas metodologías, Cascada y Ágil, hubiera permitido un análisis de primera mano más maduro.

Como contribución al desarrollo profesional de los autores, este proyecto brinda conocimiento importante y necesario de las herramientas y metodologías disponibles para que los gerentes puedan asegurar la calidad de *software*, contrastando el enfoque tradicional con el enfoque ágil. De manera más general, esto también tiene que ver con buscar la mejora continua en cualquier aspecto y tener el coraje de cuestionar el estado actual de algún procedimiento.

De mucho valor fue conocer el enfoque de Ágil en darle importancia a la comunicación e inclusión constante con el cliente en el producto que se le dará, así como mantener transparencia y comunicación con el equipo de trabajo. Esa comunicación ayuda a darse cuenta de si el proyecto tiene problemas o errores.

Como recomendaciones a futuro, será importante analizar otras variables que influyen en el éxito de proyectos de desarrollo que no se limiten al uso de una metodología específica. Esto quiere decir que factores como el tamaño del proyecto, el tipo de industria, el alcance (regional o global), entre otros, pueden tener un peso considerable sin importar si la metodología fue tradicional o Ágil.

Asimismo, investigar el uso e impacto de metodologías de desarrollo en Costa Rica puede ayudar a entender el entorno nacional y compararlo con el mundo.

Referencias

Ahamed, S. (2011). The impact of formal approaches to software quality assurance. GESJ: Computer Science and Telecommunications, 1(30).

Cirtina, L. M., Cirtina, D., & Davaris, A. (2015). Quality Assurance as Major Process of Quality Management in Projects. Applied Mechanics & Materials, 809/8101305. doi:10.4028/www.scientific.net/AMM.809-810.1305

Dawson, M., Burrell, D. N., Rahim, E., & Brewster, S. (2010). Integrating software Assurance into the software development life cycle (SDLC). Journal Of Information Systems Technology & Planning, 3(6), 49-53.

Duncan, S. s., & Peercy, D. E. (2012). Understanding Agile Concepts and the Role of Quality (Assurance). Software Quality Professional, 14(4), 13-20. El-drandaly, K. (2008). A Knowledge-Based Advisory System for Software Quality Assurance. The International Araba Journal of Information Technology, 5(3).

Fiallos, A. (2015). Mejoramiento en la productividad de software por la adaptación de un marco de desarrollo ágil. 6. Recuperado 15 Abril 2016.

García, M., Irrazabal, E., & Garzas, J. (2010). Implantación de las normas ISO/IEC 15504 E ISO/IEC 12207 con métodos ágiles y Scrum. Recuperado 31 De enero 2016.

Jain, N., & Jain, A. a. (2011). Software Development Life Cycle: A Detailed Study. International Journal Of Advanced Research In Computer Science, 2(3), 261-264.

Jamieson, J. j., & Fallah, M. m. (2012). Agile Quality Management Techniques. Software Quality Professional, 14(2), 12-21.

Owens, D., & Khazanchi, D. (2009). Software Quality Assurance. Recuperado 28 de enero 2016.

Petersen, K. A., Wohlin, C. A., Baca, D. A., Blekinge Tekniska Högskola, S. P., & Blekinge Institute of Technology, S. P. (2009). The waterfall model in large-scale development. 10Th International Conference On Product-Focused Software Process Improvement, 386.

Ivadeneira, S. (2012). Metodologías ágiles enfocadas al modelado de requerimientos. Recuperado 15 de abril 2016.

Serna, E. & Arango, F. (2011). Prueba del software: más que una fase en el ciclo de vida. Revista De Ingeniería, (35), 34-40. Recuperado 23 febrero del 2016.