CARACTERÍSTICAS DE CICLOS DE VIDA PARA PROYECTOS DE DESARROLLO DE SOFTWARE Y CRITERIOS PARA SU ELECCIÓN

Burgos Maximiliano, Gersicich Jeremías David, Gutiérrez Nehuen, Luna Rodrigo, Rugiero Franco

Abstract — El siguiente informe fue creado con dos objetivos. El primero, y haciendo referencia al título del mismo, repasar diferentes ciclos de vida para el desarrollo de software existentes, nombrando sus características principales, teniendo en cuenta que existen tantos modelos y combinaciones de los mismos como proyectos de desarrollo de software pueden existir, destacando que la correcta elección y aplicación del mismo influye directamente en los resultado del proyecto, pudiendo una mala decisión de ciclo de vida derivar en un rotundo fracaso.

El segundo objetivo está directamente ligado a la existencia de este informe. La lectura, comprensión y confección de este tipo de documentos son una parte importante del desarrollo profesional y académico de un Ingeniero, sea de la especialidad que sea, ya que los mismo son un recurso importante para la publicación de trabajos y por lo tanto la difusión del conocimiento. Comprender cómo se estructura y elaboran son habilidades incuestionables con las que un profesional del área debe contar.

Palabras Clave — ciclo de vida, iterativo, incremental, incertidumbre, riesgo.

I. INTRODUCCIÓN

"Todo esfuerzo de desarrollo de software atraviesa un 'ciclo de vida', que consiste de todas las actividades desde el momento en que la versión 1.0 de un sistema es concebido como un destello en el ojo de alguien, hasta que la versión 6.74b finalmente da su último respiro en la máquina de un cliente" (McConnell 1996).

Con este párrafo, McConnell intenta explicar de una manera entendible qué se entiende como ciclo de vida en el contexto de un proyecto de desarrollo de software.

En los 20 años, desde la publicación de ese libro hasta hoy, los principales ciclos de vida planteados en él siguen aún vigentes y, a pesar de que existen muchísimos más, siguen siendo considerados entre los más importantes.

La correcta elección de un ciclo de vida puede llevar a que el proyecto se desarrolle de la manera esperada al ser concebido, siendo sus resultados un fiel reflejo de las tareas realizadas. Esta elección va a depender de ciertas características y parámetros básicos del proyecto, siempre teniendo en cuenta que cada desarrollo es único y no existen reglas de oro en la materia. El resultado final siempre va a depender del equipo de desarrollo, el equipo de dirección del proyecto y los contextos internos y externos en los cuales esté embebido el desarrollo.

Algo que está claro es que la correcta elección de un ciclo de vida puede ayudar a mejorar los tiempos de desarrollo, la calidad del software final, disminuir la incertidumbre y sobrecostos, o mejorar la relación con el cliente. Una mala elección puede llevar a un incumplimiento total de los objetivos planteados, o a interminables jornadas de re-diseño y re-trabajo, que al mediano o largo plazo terminan desgastando a nuestro cliente y a los propios integrante de los equipos.

II. DESARROLLO

A. Empezar por las bases

Al igual que en el libro de McConnell o diferentes estudios de la materia, como el paper de Nayan B. Ruparelia, tomaremos al **ciclo de vida en cascada** como la base para iniciar el análisis de los demás. Como ambos plantean, más allá de la eficacia o no de este modelo, las actividades que abarca estarán más o menos presentes en cualquier otro ciclo de vida utilizado para el desarrollo de software (salvo contadas excepciones), por lo cual es un punto de referencia inicial muy interesante.

Ya en 1996, con foco en el título de su libro *Rapid development* y sin tener en cuenta las corrientes ágiles que se encuentran implantadas actualmente, McConnell plantea que los principales problemas de este ciclo de vida

son su estructura por demás rígida (se compone con un número fijo de actividades, y cada una debe ser completada en su totalidad para poder avanzar a la siguiente) que lo hace extenderse en el tiempo, así como la innumerable cantidad de documentos que se producen como salidas de cada una de sus actividades. También expresa como falencias del modelo la ansiedad que le genera al cliente el querer ver su producto funcionando (lo cual es solo posible al completar la cascada), el traslado de esta ansiedad como presión al equipo de desarrollo, y lo costoso o hasta imposible que puede llegar a ser un re-trabajo en caso de haber omitido o mal logrado algún tipo de información o tarea en las etapas anteriores. Estos problemas hacen del modelo en cascada un ciclo de vida poco conveniente a utilizar bajo condiciones reales, aunque teóricamente o hasta encuadrado en ciertos parámetros ideales, un proyecto de software realizado con este ciclo de vida pueda producir software altamente confiable y con las bases necesarias para poder seguir creciendo.

B. Tomar lo bueno y flexibilizar

A pesar de sus contraindicaciones, el sistema en cascada termina dejando una base para el desarrollo del resto de los ciclos de vida propuestos: los requerimientos de software deben ser definidos y analizados previo a realizar cualquier esfuerzo de diseño o implementación.

Con el tiempo fueron surgiendo diferentes alternativas al modelo de cascada puro, algunas de las cuales son planteadas en el libro de McConnell y otras que aparecen décadas más adelante.

Lo que persiguen estas alternativas es disminuir el impacto de lo impredecible y del riesgo inherente de cualquier proyecto de desarrollo de software. Para esto, McConnell nombra diferentes modificaciones al cascada original como *Sashimi*, planteado por Peter DeGrace, para lograr que diferentes actividades puedan ser superpuestas en el tiempo, logrando así cierta retroalimentación o revisión de las salidas de cada una, y así evitar que errores se propaguen sin visibilidad hasta el final del ciclo.

Otra modificación propuesta en el mismo sentido es la realizada por Winston Royce en 1970, quien plantea mayor flexibilidad a la hora de retroalimentar a la etapa anterior, así como también sobre los documentos que se generan como salida de cada una, dotando al modelo de cierta iteratividad, permitiendo por ejemplo que haya feedback entre las tareas de Diseño-Requerimiento y Validación-Diseño. De esta forma se logra que, cosas que se hayan escapado anteriormente, se puedan incorporar y arreglar antes de dar por finalizado el ciclo entero.

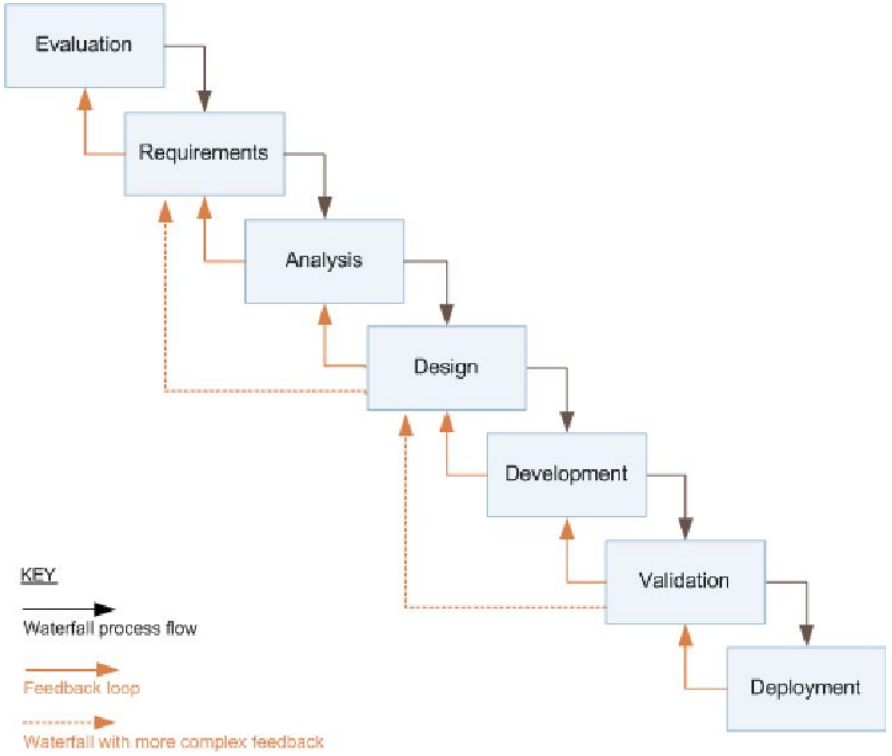

Figura 1: Modelo en cascada con retroalimentación de Royce

Otra variante del ciclo de vida en cascada es el modelo incremental. Este modelo consisten en la implementación del modelo en cascada pero en sucesivas iteraciones; en vez de completar cada actividad para el 100% del software a desarrollar, se realiza de a porciones o incrementos pequeños. De esta manera cada actividad e iteración retroalimenta a la siguiente. Con este modelo se logra, por ejemplo, disminuir la ansiedad del cliente al tener parte del software funcionando al final de cada iteración, involucrandolo en el relevamiento de requerimientos de la siguiente.

De igual manera, realizar el desarrollo de manera incremental permite tener una noción de los cambios introducidos en cada versión, así como también aislar en alguna de ellas posibles errores que puedan surgir.

Este tipo de ciclo de vida también es utilizado con modelos diferentes al de cascada, como por ejemplo el Proceso Unificado, creado por Rational Software y luego comprado por IBM, que también utiliza el concepto de iteraciones e incrementos para ir generando los diferentes modelos que lo componen e ir refinando en las subsecuentes iteraciones.

McConnell plantea adicionalmente en su libro una versión modificada del modelo en cascada, en el cual se intenta, antes de empezar con el proceso en cascada en sí mismo, disminuir el riesgo y la incertidumbre que surge de tener que analizar los requerimientos y definir la arquitectura del sistema completamente al comenzar el proyecto (al menos en la medida de lo posible), mediante la utilización de una *espiral de reducción de riesgo* (técnica que se explicará más adelante).

C. Reducción del Riesgo

Como se mencionó en el párrafo anterior, McConnell

plantea en su libro la posibilidad de aplicar una modificación del modelo en cascada mediante la utilización de una “espiral de reducción de riesgo”.

En sí misma, la espiral es considerada como un formato de ciclo de vida más que puede ser utilizado para el desarrollo de software, el cual surge de una modificación propuesta al modelo en cascada, realizada por Boehm en 1986.

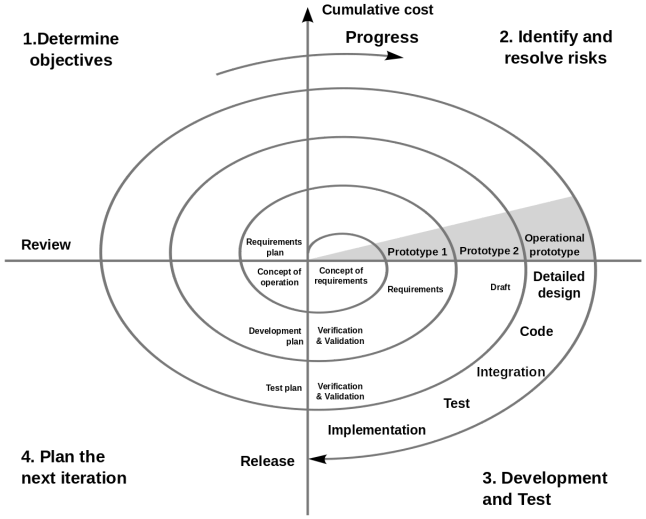

Figura 2: Espiral de Boehm.

La propuesta de Boehm se basa principalmente en que las etapas de diseño del sistema en cascada pueden, en ciertos contextos, llegar a ser innecesarias, y que cierta moderación es necesaria a la hora de realizar el desarrollo, a fin de poder tener una visión a más largo plazo, pensando en la reutilización del código e identificando posibles riesgos y problemas en etapas más tempranas del desarrollo.

Esta propuesta significa un cambio de paradigma a la hora de pensar el desarrollo de software, ya que se tiene más en cuenta los riesgos que los requerimientos en sí mismos, con el objetivo de minimizar los costos que estos puedan acarrear.

Para llevarlo adelante se subdivide el proyecto en varios mini proyectos, intentando atacar en cada uno de estos los mayores riesgos hasta que finalmente no quede ninguno.

En este caso, las definiciones de riesgo dependen mucho del contexto. Estas pueden ir desde problemas de performance hasta problemas intrínsecos de las tecnologías utilizadas.

El modelo en espiral también se basa en iteraciones incrementales, donde cada iteración lleva al proyecto a un estadio mayor.

Cada iteración se encuentra se encuentra marcada por las siguientes 6 tareas:

1. Determinar objetivos, alternativas y limitaciones.
2. Identificar y resolver riesgos.
3. Evaluar alternativas.
4. Desarrollar los productos o entregables para esa

iteración y validar los (prototipos o software final).
5. Planificar la próxima iteración.
6. Comprometerse a una modalidad para la próxima iteración.

III. Conclusión

A. Cómo decidir qué ciclo de vida utilizar

Al elegir la temática y con algo de los conocimientos previamente disponibles, nuestra primera impresión era que, luego de leer un poco sobre las diferentes alternativas, iba a ser fácil decantarse por una opción. Así como hubiera sido imposible abarcar todos los ciclos de vida existentes, es que termina siendo casi imposible indicar con total seguridad que ciclo de vida es mejor que otro, cual debe utilizarse por sobre, como si existiera una regla general. La variedad de los mismos decanta de una realidad: **ningún proyecto de desarrollo de software es exactamente igual a otro**.

La mejor manera de decidir qué ciclo de vida a utilizar es entendiendo cuáles son las características de cada uno, sus bondades y deficiencias, tomando conciencia de que no son modelos fijos, y que de cada uno se puede tomar las herramientas y características que mejor encajen en nuestras necesidades. Es necesario evaluar los niveles de riesgo, la claridad y cantidad de requerimientos, los conocimientos técnicos con los que cuenta el equipo de desarrollo, los plazos que se tienen para ejecutar el proyecto y la posibilidad de poder validar los resultados del trabajo con el cliente antes de elegir un ciclo de vida.

Finalmente, se debe tener en cuenta que, a pesar de cualquier análisis que se pueda hacer y de haber elegido un ciclo de vida que dio buenos resultados en proyectos con características similares al que se está por emprender, si el mismo es ejecutado de una manera pobre, los resultados pueden ser igual de malos a como hubieran sido si no se hubiera elegido ningún modelo y nos hubiéramos puesto a escribir líneas de código como si no hubiera mañana.

B. Tomando la decisión

La realidad es que, por más complejas que sean las variables para la elección del ciclo de vida, en la actualidad, los tiempos de desarrollo se hacen cada vez más cortos, todo se necesita para ahora y equivocarse puede llevar a muchas horas de retrabajo y por lo tanto altos costos. La ***elección de un ciclo de vida iterativo incremental*** que ayude a reducir el riesgo, como el Espiral de Boehm, se configuran en una opción segura, que se complementa de manera correcta con las más que nunca vigentes corrientes de desarrollo Ágil, permitiendo tener software funcionando, con resultados visibles, evitando tener que hacer grandes re-trabajos y validando el avance junto con

el cliente en tiempos relativamente cortos y que además disminuyen los riesgos e incertidumbre que pueden afectar en las fases finales de un proyecto de desarrollo de software realizado con un ciclo de vida en cascada.

Anexo

Elección del template de Informe Técnico

Para el formato de este informe, no se utilizó ninguna plantilla base, si no que se usaron un compendio de diferentes formatos de papers que se utilizaron como fuentes para este trabajo, así como también diferentes partes rescatadas de diversas templates en vez de utilizar una específica. Dicha información fue sacada de las siguientes páginas:

[1] **Nayan B. Ruparelia (2010):** *"Software Development Lifecycle Models"*. En *ACM SIGSOFT Software Engineering Notes archive, Volume 35 Issue 3, May 2010, Pages 8-13.* ACM New York, NY, USA; https://inf.ufes.br/~monalessa/PaginaMonalessa-NEMO/ES_Mestrado/Artigos/ModelosCicloVida-Ruparelia-ACM2010.pdf

[2] **Lucecita Bells:** *"Informe técnico"*; Academia.edu; https://www.academia.edu/5091078/INFORME_T%C3%89CNICO_LA_ESTRUCTURA_DEL_INFORME_T%C3%89CNICO_EST%C3%81_FORMADA_POR_La_parte_inicial

[3] *"Cómo estructurar un Informe Técnico"*; Universidad de Cantabria; https://ocw.unican.es/pluginfile.php/1408/course/section/1805/tema10-comoEstructurarUnInformeTecnico.pdf

[4] **Valerio Ccori:** *"Modelo de Informe Técnico";* Academia.edu; https://www.academia.edu/24567206/Modelo_de_Informe_Tecnico

[5] **Assessment Medical Research Paper;** Template.net; https://images.template.net/wp-content/uploads/2017/06/Assessment-Medical-Research-Paper.zip

[6] **Sample Medical Research Paper;** Template.net; https://images.template.net/wp-content/uploads/2017/06/Sample-Medical-Research-Paper.zip

[7] **IEEE conference templates;** IEEE; https://www.ieee.org/content/dam/ieee-org/ieee/web/org/conferences/Conference-template-A4.doc

Bibliografía

[1] **McConnell, Steve (1996):** *"Rapid development: taming wild software schedules"*. Microsoft Press, Redmond, Washington. Capítulo 7.

[2] **Nayan B. Ruparelia (2010):** *"Software Development Lifecycle Models"*. En *ACM SIGSOFT Software Engineering Notes archive, Volume 35 Issue 3, May 2010, Pages 8-13.* ACM New York, NY, USA.

[3] **Boehm, Barry W. (1986):** *"A spiral model of software development and enhancement"*. In ACM SigSoft Software Engineering Notes, Vol. II, No. 4, 1986, pp 22-42.