Investigación y Tecnología Vol. 4. Nro. 1: pag.128 – pag.132. Octubre 2016. ISSN 2306-0522

Método para el Orden de Implementación de Software

Method for Software Deployment Order

*** Ariel Richard Condori Rodríguez**

Resumen

En este artículo se explica un método para el orden implementación de software. Muchos profesionales y egresados inician en el desarrollo de software con la duda de encontrar el inicio de la implementación, el método propone una solución a este problema.

El método relaciona dos grandes campos enigmáticos, la teórica y la práctica. La solución se basa en los diagramas de clases y de forma equivalente en un modelo relacional. Para el lector no familiarizado con los conceptos usados en el presente artículo, se expone respaldo bibliográfico para su consulta. Se detalla el fundamento del método y la aplicación del método.

Palabras clave: Orden de implementación, dependencia (Acoplamiento), Iteración, Diagrama de clases, Modelo relacional.

Abstract

This article describes a method for order software implementation is explained. Many professionals and graduates start in software development with doubt find the start of implementation, the method proposes a solution to this problem.

The method relates two large enigmatic fields, theoretical and practical. The solution is based on class diagrams and equivalently in a relational model. For unfamiliar with the concepts used in this article, bibliographic reader support for consultation exposed. the foundation of the method and the application of the method is detailed.

***Keywords**: Order of implementation, dependence (Coupling), Iteration, Class Diagram, Relational Model.*

Introducción

En situaciones el joven profesional inicia en el campo de desarrollo de software, emprende un proyecto, si en su ambiente existe gente con experiencia, aprenderá los métodos para el desarrollo y su desempeño ira mejorando con el tiempo. En otra situación, el joven profesional empezará de forma individual o en un equipo que está en sus primeros pasos en el desarrollo de software, lo más probable es que sus estimaciones y planificaciones no cumplan con las expectativas del equipo. Entonces nace la pregunta, "¿Por dónde empezamos a desarrollar?".

Después del análisis y diseño de software continuamos con la implementación. En la implementación es donde este método es aplicado para poder tener un orden de implementación y posteriormente marcar prioridades de casos de uso o historias de usuario, terminando con una estimación del tiempo de desarrollo.

Craig Larman expone una idea fundamental para el orden de implementación en su libro UML y Patrones, el cual es usado como base para la elaboración del método.

Las metodologías existentes hacen énfasis en el término "iterativo e incremental", ya sea una metodología pesada o ágil. De cualquier manera, en el desarrollo de software es evidente que este término se cumple, aunque no de la mejor forma.

La idea fundamental en la implementación, es definir posibles iteraciones y en cada iteración se debe elegir un conjunto de casos de uso, el criterio puede ser la complejidad y/o prioridad. Para ello debemos seleccionar que clases debemos implementar. (Ivar Jacobson, Grady Booch y James Rumbaugh, 2000).

En las metodologías agiles se tiene una pila de historias de usuario y sus tareas, y se realiza su correspondiente estimación. El principio es que el cliente escoge las historias de mayor prioridad y estas historias estarán dentro de la siguiente iteración. Las historias dirigen el desarrollo. (Beck, 1999).
Pero no se tiene un método para proveer un orden de implementación y niveles de iteración.

El concepto fundamental para asociar clases es que cada una de ellas tiene una responsabilidad que cumplir y en colaboración se debe cumplir con los requerimientos del sistema.

Métodos

El método ayuda a poder visualizar el progreso de desarrollo de software de forma conjunta entre clases, casos de uso o historias de usuario (depende de lo que decida usar).

El objetivo del método es realizar un posible orden de implementación para tener como base el inicio en el desarrollo, este orden puede ser modificado por criterios que el grupo o persona vea conveniente.

Algoritmo

El proceso del algoritmo es contar las dependencias de cada clase, calcular niveles de dependencia y finalmente representar este orden de implementación mediante algún diagrama, tarjetas CRC u otro medio.

Pasos:

1. Sumar las dependencias que posee cada clase y escribirlo cerca de la clase correspondiente

2. Crear Niveles: escribir el nivel delante de la dependencia de cada clase separado por un "." (punto) < nivel . dependencia > ejemplo. 1.x – 2.x – 3.x
Para cada clase desde la menor dependencia hasta la clase con mayor dependencia

Si la clase no tiene dependientes, entonces
El nivel de la clase es 1 (uno)
Si la clase tiene muchos dependientes, entonces
Seleccionar el de mayor nivel e incrementarlo en 1 (uno).

3. Caso especial:

Si la clase tiene solo una clase dependiente, y la lógica de negocio indica que las dos clases se colaboran en un mismo instante, entonces puede mantener el mismo nivel de la clase que depende, y ajustar el número de dependencia de la clase.

Este caso es aplicable a relaciones de tipo composición y agregación.

Mejoras en Tarjetas CRC

Todos los integrantes del equipo sugieren nombres para las clases y se van clasificando cuales son fundamentales para el proyecto, se mantiene pendiente las clases que se tienen dudas (Beck, 1999). Ahora con el método se puede agregar a las tarjetas su respectivo nivel y dependencia, y ordenarlo por niveles e iniciar la implementación.

Mejoras en los Casos de Uso

Una forma de representar el progreso en los casos de uso, es agregar el número de nivel de implementación al caso de uso y marcarlo con un color distintivo cuando el caso de uso termina siendo implementado

Fundamento del Método

Craig Larman indica: "Es necesario implementar las clases (y, en teoría, probarlas totalmente de manera individual), de la menos acoplada[26] a la más acoplada. Por ejemplo, las posibles primeras clases para implementar son *Pago* o *EspecificacióndeProducto*. Vienen después las clases que dependen únicamente de las implementaciones anteriores, *CatalogodeProducto* o *VentasLineadeProducto*." (Larman, 2002). Pero este fundamento carece de niveles. El método indica niveles de implementación, para posteriormente planificar posibles iteraciones.

Resultados

Con el concepto propuesto por Craig Larman, tendremos el siguiente orden de implementación. A diferencia de este, el método propuesto indica niveles de implementación, ahora podremos comparar los resultados que provee el método. Para la comparación, usaremos el ejemplo empleado en el libro de UML y Patrones de Craig Larman.

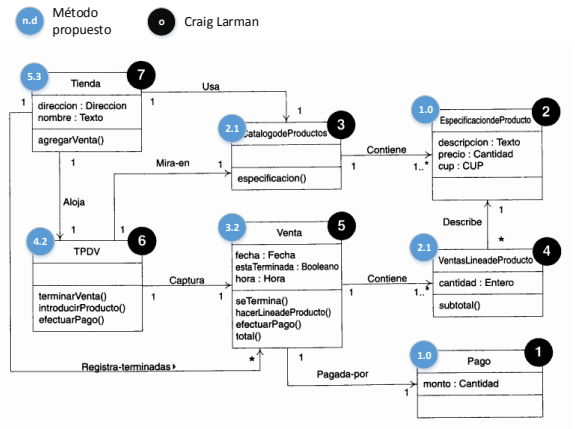

Ilustración 27. Posible Orden de Implementación (Larman, 2002) y Método propuesto

[26] Acoplamiento: Dependencia entre elementos (generalmente tipos, clases y subsistemas), normalmente debida a la colaboración entre ellos para prestar un servicio.

Orden Craig Larman	Método propuesto	Clase
1	Nivel 1.0	Pago
2	Nivel 1.0	EspecificaciondeProducto
3	Nivel 2.1	CatalogodeProductos
4	Nivel 2.1	VentasLineadeProducto
5	Nivel 3.2	Venta
6	Nivel 4.2	TPDV
7	Nivel 5.3	Tienta

Si se agrega una nueva clase que depende de "Tienda" el concepto de orden de implementación según acoplamiento, pierde coherencia en el orden, en cambio el método propuesto considera esta situación y responde a este cambio.

El método parece ser no tan útil con algo pequeño, pero supongamos un ambiente en el siguiente módulo, en esta situación el método es de gran ayuda.

Discusión

Existen diferentes ambientes para el desarrollo de software, tanto en productos a medida o estándar, por lo que cada ambiente puede seleccionar el método que más se adecue para la implementación, por ejemplo, "issues", "task", "user stories" dirigen la implementación, o también como "use case". El método propuesto puede ser adaptado desde un nivel macro, para luego iniciar con la implementación.

Puede ser una desventaja como también una ventaja, El método es empleado en la fase de análisis respaldándose en uno de los siguientes diagramas: diagrama del dominio, diagrama de clases o esquema de base de datos.

Conclusiones

El método propone un posible orden de implementación sobre las clases, aclarar que esta implementación la dirige los casos de uso o historias de usuario con el factor de prioridad.

A partir del posible orden de implementación de las clases, es posible establecer la complejidad de cada una de ellas y dar como resultado un tiempo estimado de implementación. El método es de gran ayuda cuando se inicia en desarrollo, con la práctica, el método es usado de forma implícita.

Es posible también usar el método en base a un esquema de base de datos y lograr tener un esquema de dependencia y jerarquía de niveles para su futura implementación.

Agradecimientos

Un agradecimiento especial a mi Docente en años de universidad y un gran amigo, que lo estimo mucho, Ing. Abdón Bustos Condori. (Q.E.P.D). Gracias por sus palabras que siempre impulsaban a estar dentro del saber. Gracias por ser un gran amigo.

Docente Ing. Julio Cesar Bermúdez, gracias por su tiempo para evaluar el método e impulsar a compartirlo.
A Lic. Waldo Mendoza, que con sus preguntas encontré una visión amplia sobre el método. Gracias.

Referencias

Beck, K. (1999). *Extreme Programming Explained.* Embrace Change.

Ivar Jacobson, Grady Booch y James Rumbaugh. (2000). *El Proceso Unificado de Desarrollo de Software.* Addison Wesley.

Larman, C. (2002). *UML y Patrones Introducción al Análisis y Diseño Orientado a Objetos.* Prentice Hall.

Presentado: La Paz, 16 de septiembre de 2016.

Aceptado: La Paz, 10 de Octubre de 2016