Una aproximación a un Modelo Concurrente de Ciclo de Vida Software con Calidad Total

Alejandro Estayno, Marcelo Estayno, Alicia Mon
{estayno, mestayno, aliciamon}@fibertel.com.ar
G.I.S. – UNLaM[1]
Argentina

Summary

The modeling of the Software Process is a guideline for the organization of the activities which involve all the development stages for the resolution of the problems that the software development presents and generates as it evolves. The modeling of the product life cycle becomes more and more complex as the technological and methodological development and the organization of the work becomes more complex.

The purpose of the Modelo Concurrente de Ciclo de Vida hereby explained is to include in the software development the Total Quality and Quality Management concepts based on the detention of the process in case of failures and through the continuous improvement. These are key elements introduced by the Toyota Model for the organization of industrial processes. Moreover, the software development Agile Methods has been included. These methods tend to converge in the concurrent development and in the conditions for the structuring which is proposed in this Model.

Resumen

La modelización del Proceso Software constituye un marco de referencia para la organización de las actividades que involucran todas las etapas del desarrollo, tendente a la resolución de problemas que el propio desarrollo de software plantea y genera a medida que evoluciona. La modelización del ciclo de vida del producto software se hace mas compleja a medida que el desarrollo tecnológico, metodológico y, fundamentalmente la organización del trabajo aumenta su complejidad.

En el Modelo Concurrente de Ciclo de Vida (MCCV) de software que se expone en el presente artículo, se propone incorporar al desarrollo de software los conceptos del Calidad Total o Quality Management basados en la detención del proceso ante la presencia de fallos y a través de la mejora continua, elementos claves introducidos por el Modelo Toyotista de organización de procesos industriales. Asimismo, se han incorporado principios de los Métodos Ágiles de desarrollo software que, tienden a confluir en el desarrollo concurrente así como en las condiciones de ordenamiento propuesto en el presente Modelo.

Palabras Claves: Proceso Software; Modelos de Ciclo de Vida; Calidad Total

[1] Grupo de Ingeniería de Software. Secretaría de Posgrado. Departamento de Ingeniería e Investigaciones Tecnológicas. Universidad Nacional de La Matanza.

1. Introducción

La Ingeniería del Software, al igual que otras ingenierías más tradicionales, como la ingeniería eléctrica o industrial, se divide en dos grandes áreas de actividad: la ingeniería del producto y la ingeniería del proceso. Hasta la década de los 80, el área de actividad predominante fue la ingeniería del producto, debido principalmente a dos razones: la juventud de la disciplina, y el relativo desconocimiento de cómo desarrollar, desde el punto de vista puramente tecnológico, un software efectivo, fiable y de bajo costo.

Sin embargo, el desarrollo conceptual del proceso software no es ajeno a la evolución de las formas de organización en la producción en general. El surgimiento de Metodologías Ágiles en el desarrollo de software da cuenta de ello, al introducir formas de organización provenientes del modelo japonés, que se encuentran inmersas en los ejes organizacionales de la producción industrial en la actualidad, tales como el desarrollo modular, la flexibilidad en los procesos, la importancia de los conocimientos, o la programación por pares, tendencias todas a quebrar las fronteras en los compartimentos estancos de los equipos de desarrollo. Así, la detención del proceso ante la presencia de fallos y la transferencia de conocimiento son elementos organizativos centrales del proceso software que permitirían mejorar la representación del ciclo de vida del producto software.

El área de estudio sobre proceso software se reconoce actualmente como un factor crucial, ya que su propósito es proveer el conjunto de actividades que permiten, a partir de la expresión de una necesidad, realizada por un cliente o usuario, derivar un sistema software que satisfaga dicha necesidad. La importancia del proceso software en el contexto de la ingeniería del software está respaldada por el gran número de trabajos de investigación que se han realizado hasta la fecha. Los resultados de dichos trabajos de investigación han sido numerosos estándares o Modelos de proceso, como el IEEE1974 [1] o el ISO/IEC 12207[2] [3], modelos de madurez como el ISO 15504 [4] o el CMMI [5] y los diferentes Modelos de Ciclos de Vida [6] que representan las transformaciones del producto software a medida que se va desarrollando a través del proceso.

Con el objetivo de mejorar la definición del proceso software, tanto desde el punto de vista de cada actividad particular a realizar, como desde el punto de vista de la organización en su conjunto, introducimos el Modelo Concurrente de Ciclo de Vida software (MCCV) [7] y la aplicación de criterios específicos de organización de la producción de modo tal que nos permita incorporar aquellos elementos que pueden analizarse en el entorno y las especificidades del desarrollo de software en cuanto a las formas de organización del trabajo .

2. El Modelo Toyotista

El modelo industrial japonés desarrollado y aplicado por Toyota, conocido habitualmente como *modelo toyotista* [8], introduce una serie de innovaciones en las formas organizacionales de la industria basadas en la eliminación de tiempos muertos, la desaparición del stock de productos por medio de la implementación del j*ust in time*, produciendo exclusivamente sobre pedido. A su vez, desarrolla las técnicas innovadoras del Total Quality Management (TQM) [9], basadas en la detención del proceso de producción ante la presencia de fallos y la implementación de la *mejora continua*, haciendo que el producto fluya a instancias del cliente intentando tender a la producción con "cero defectos".

La implementación de estos principios del Modelo están determinados por la *pluriespecialización* de los operadores de planta, es decir, en la ampliación de sus conocimientos y su relación participativa en la toma de decisiones y en todas las instancias de la producción, así como la especi-

ficación de principios y métodos iterativos, auto-organizativos e interdependientes en un patrón de ciclos de corta duración.

En este sentido, la producción requiere de una organización del trabajo ágil, capaz de mantener un alto nivel de adaptabilidad y flexibilidad en el empleo del trabajo y las capacidades.

Las principales innovaciones que aporta el modelo organizativo japonés, destaca las siguientes características [8]:

- **Técnicas de planificación**

Una revolución en las técnicas de planificación y optimización de la puesta en marcha de la producción, que consiste en una inversión de las reglas tradicionales, en lugar de que la fabricación se desarrolle "en cadena" de arriba hacia abajo, se hace de abajo hacia arriba, partiendo de los pedidos y de los productos ya vendidos.

Como mencionamos anteriormente un ejemplo representativo de este tipo de organización se implementó en la producción automotriz, comenzando su línea de planificación de la producción por el último puesto de ensamblaje en la organización, que es el punto más próximo al producto terminado. A partir de los productos vendidos o entregados, se inicia un pedido de reposición sobre la cantidad exactamente vendida, sin necesidad de generar un almacenamiento de una cantidad de productos a la espera de ser entregados.

- **Flujo de información**

Esta inversión en el "sentido" general de las instrucciones de producción, requiere de una fuente de información precisa y ajustada a la necesidad de producción exacta en cada puesto. La clave del método consiste en establecer paralelamente al desarrollo de los flujos reales de producción (de arriba a abajo), un flujo de información invertido de abajo hacia arriba, emitiendo cada puesto corriente abajo una instrucción destinada al puesto corriente arriba inmediatamente anterior. Esta instrucción consiste en el pedido de la cantidad y la especificación exacta de las unidades necesarias al puesto corriente arriba para ejecutar su propio pedido.

Desde abajo, la serie de pedidos va de puesto en puesto y remonta corriente arriba, de tal manera que en un momento dado, en el departamento que se considere sólo hay en producción la cantidad de unidades exactamente necesaria, cumpliéndose el principio de "*cero existencias*", solo se está produciendo lo estrictamente solicitado.

- **Planificación**

Esta modificación en la información permite descentralizar una parte de las tareas de Planificación, especializado y ajeno a los propios productores y confiar la responsabilidad de ellas a los jefes de cada equipo.

- **Autonomía en cada puesto**

Todo el sistema de circulación de la información se lleva a cabo mediante un doble movimiento, de inscripción de pedido hacia arriba, y de productos ya producidos o desarrollados a partir de los pedidos, hacia abajo. Esto genera la necesidad de la planificación en cada puesto de trabajo, asegurando la autonomía en la toma de decisiones y el conocimiento de los trabajadores en cada puesto.

Estos tres elementos indican claramente que la innovación es sólo de organización y conceptual, sin que intervenga lo tecnológico y que responde a una organización con una inmediata capacidad de respuesta al mercado que aporta producción y montaje modular siguiendo de cerca estrictamente el flujo de los pedidos en cantidad y calidad, permitiendo de este modo, obtener una varie-

dad de productos sobre la base de componentes elementales capaces de adaptarse a las diferentes combinaciones de productos pedidos.

- **Gestión de la Calidad en los propios puestos de desarrollo**

Integrar las tareas de Control de Calidad de los productos a las tareas de desarrollo. Reasociar las tareas de ejecución, programación o control de calidad, incorporando los conocimientos dados en la pluriespecialización y polivalencia de los profesionales. La reintroducción de las tareas de Control de Calidad a los puestos de trabajo tiene por objeto que los propios puestos de desarrollo se hagan cargo de la calidad de los productos. En este sentido, la pluriespecialización de los operadores es lo que hace posible que estos se encarguen de la gestión de calidad y de algunas dimensiones de la planificación. Así también, la ejecución de las tareas variadas (fabricación, reparación, control de calidad y planificación) alimenta y enriquece permanentemente la "polivalencia" y los conocimientos prácticos de los operadores, generando un proceso de "aprendizaje dinámico".

El método japonés de linealización consiste en concebir instalaciones y una cierta distribución del espacio físico de forma tal que permita la movilización de los trabajadores pluriespecializados, y el cálculo permanente de los estándares de operación que se les asigna a cada uno. En su conjunto, la linealización constituye un sistema de recomendaciones prácticas basadas en una concepción de la producción ampliamente renovada, que se centra en movilizar los recursos de "tiempos y movimientos" en organizaciones e implantaciones flexibles.

El dispositivo destinado al mejoramiento de la calidad de los productos, introduciendo en el nivel de los propios procesos operatorios dispositivos de llamado que tienen por objeto prevenir el error, hacerlo casi imposible, generando el paro de una máquina o un proceso en cuanto se presenta un riesgo de defecto de fabricación o de desarrollo. Se trata de dispositivos que pueden adaptarse a los equipos y a las herramientas tendiendo a "cero defectos" [8].

Este tratamiento de la calidad implica quedarse con la manera de ejecutar una tarea cualquiera, que presente la mayor garantía en cuanto a la calidad del producto, en vez de buscar la rapidez de ejecución. Sobre esta base los japoneses han elaborado progresivamente un impresionante conjunto de herramientas de Gestión de Calidad que culminan en las diferentes técnicas llamadas de "Calidad Total" [10].

Procedimientos que permiten hacer literalmente "visible" el desarrollo del proceso de producción al permitir una visualización de cada uno de los acontecimientos susceptibles de producirse como puede ser el exceso o insuficiencia de existencias de "requisitos" con relación a los pedidos, interrupción o disminución de la velocidad del flujo. Esta disponibilidad física y práctica, lo que genera básicamente es incitar a los trabajadores a que no duden en detener el proceso de producción, es el mejor medio para asegurarse de que se hará todo para eliminar prontamente las anomalías. Este control visual y la incitación al autocontrol son características muy claras del método japonés.

3. El Modelo de Ciclo de Vida Concurrente MCCV-1205

Dentro de las actividades asociadas a las fases de un ciclo de vida de software no se puede pasar de un estado a otro, ni siquiera ejecutar un proceso por simple y mínimo que fuera dentro del Ciclo de Vida sin algún tipo de conocimiento. El proceso de adquisición del conocimiento no es secuencial y menos aún lineal. El planteo básico es el siguiente: cada porción del conocimiento necesaria puede ser transmitido de un proceso -subproceso- a otro, para lograr pasar de un estado -intermedio- a otro. El ciclo de vida de transformación del producto software no se expresa como un

proceso secuencial, es un proceso concurrente y no hay posibilidades de determinación para las coordenadas temporales de los pasajes de conocimiento.

En este sentido y para ejemplificar con una parte del proceso, si alguna de las subetapas de la etapa de análisis comienza, es porque se tiene el conocimiento, concreto y necesario para darle inicio. El conocimiento es portado, por ejemplo, por un entregable, como input del proceso asociado a la subetapa. Este entregable también representa el momento en que están dadas las condiciones para comenzar a ejecutar el proceso, por medio del cual se cambia de estado, progresando en la dinámica del ciclo de vida del producto.

3.1. La concurrencia en el proceso

En un proceso de construcción de la solución en términos de los recorridos hacia la solución definitiva del problema, su dinámica no luce como un proceso de derrotero continuo. Haciendo una analogía con el proceso mental disparado ante un problema emergente a solucionar. Al instante de conocer el problema que se resolverá con la implementación de una solución concretada en un sistema software, se establecen una confluencia caótica de escenarios y actos mentales.

Una alternativa posible, para la resolución del problema, para alguien con cierta experiencia en el desarrollo de aplicaciones software podría ser:

- primera versión psíquica de la interfase software
- conjunto mínimo de funcionalidades básicas
- ciertas funcionalidades esenciales
- algunos problemas de las funcionalidades
- algunas soluciones a un subconjunto de los problemas anteriores
- algún integrante del team a quien consultar problemas a resolver
- analogías con sistemas conocidos
- posibles reutilizaciones de soluciones implementadas
- primera visión de la estrategia tecnológica
- universo de usuarios
- imagen de un layout posible del team project
- alguna circunstancia en el escenario de desarrollo de la solución
- estimación bruta del tiempo
- ...

Aunque en términos clásicos se esté en medio de un estadio típico de captura de requerimientos, concurrentemente se han establecido conexiones y se ha logrado un avance en el recorrido hacia la solución del problema. En estos caminos alternativos, se han disparado procesos que anticipan ciertas fases del ciclo de vida de desarrollo de un producto software como análisis, prototipado, diseño, test y codificación. Aún más, este proceso se hace constante y tiene como único límite la finalización del ciclo de vida.

Dejando a un lado, por el momento, la concurrencia de avances sobre las fases del ciclo de vida, surge la cuestión sobre cuáles son las condiciones para que cada uno de ellos avance o se detenga. Es por ello que, la incorporación del mecanismo de semáforos, permite incorporar al modelo de ciclo de vida propuesto, una herramienta teórica para facilitar la detención o continuación del proceso.

3.2. Los semáforos de Dijkstra

Los *semáforos,* concebidos por Edsger Wybe Dijkstra [11] [12], constituyen una herramienta de sincronización que restringe o permite el acceso a recursos, que en el caso del MCCV, los recursos o entregables están conformados por el conocimiento.

De acuerdo a las definiciones presentadas en Operating System Concepts [12 bis], un semáforo **S** es una variable entera a la que, una vez que se le ha asignado un valor inicial, sólo puede accederse a través de dos operaciones atómicas estándar: espera (wait) y señal (signal). Estas operaciones se llamaban originalmente **P** (para espera; del holandés *proberen*, probar) y **V** (para señal; del holandés *verhogen*, incrementar).

El valor de una variable del tipo semáforo es el número de unidades del recurso (en nuestro caso conocimiento/s) que está/n disponible/s. La operación P detiene la ejecución hasta que hay un recurso disponible, en cuyo caso lo reclama inmediatamente y sigue su ejecución normal. V es la operación inversa; hace disponible un recurso común, y permite que uno de los procesos suspendidos por la falta de ese recurso, prosiga con su ejecución. La operación *Inicia* se utiliza para inicializar el semáforo antes de que se hagan peticiones sobre él.

Las definiciones clásicas de espera y señal son:

```
Espera(S): while S = 0 do nada;
           S:= S - 1;

Señal(S): S:= S + 1;
```

Las modificaciones del valor entero de los semáforos en las operaciones Espera y Señal se deben ejecutar de forma indivisible, es decir, mientras un proceso modifica el valor del semáforo, ningún otro proceso puede modificar simultáneamente el valor de ese mismo semáforo. En el caso de Espera(S) la prueba del valor entero de S (S = 0) y su posible modificación (S := S - 1), también deben ejecutarse sin interrupción.

3.3. Los semáforos y el modelo

En el MCCV cada fase de un ciclo de vida se comporta como un dispositivo receptor y transmisor de conocimiento sobre *cómo* debe/n seguir otra/s fase/s. De este modo, cada instancia del producto, desarrollada en una fase del ciclo de vida, podría pasar a otra fase del desarrollo, cualquiera que sea, solo si tiene una aceptación de que todo lo necesario para esa parte ha sido completada y si la fase que la requiera para iniciar su ejecución cuenta con los conocimientos necesarios para comenzar dicha instancia, y cada fase continúe trabajando en forma concurrente con otra en la misma parte del producto.

La sucesión de instancias está determinada por las condiciones para que cada proceso, avance o se detenga. Para ello, el MCCV incorpora el mecanismo de semáforos, para instanciar los permi-

sos que cada fase tiene para entregar y/o para recibir una parte del producto.

Los semáforos, en este caso resignificados en una herramienta conceptual de sincronización, son utilizados para modelar el flujo en el tiempo de las actividades asociadas a cada fase, que cooperan entre sí transfiriéndose el conocimiento necesario para proseguir. Cada actividad asociada a cada fase de un ciclo de vida se comporta como un dispositivo receptor y transmisor de conocimiento sobre cómo debe /n seguir otra /s fase /s.

Por ejemplo, si suponemos un conjunto extremadamente clásico de fases: Requerimientos, Análisis, Diseño, Implementación y Pruebas, bajo las concepciones de un modelo secuencial clásico, la tecnología a aplicar en el desarrollo de una solución software es una decisión de diseño. Pero si en la etapa de requerimientos, con los primeros conceptos, más allá de las técnicas empleadas para la captura de requisitos, se obtiene información básica sobre la tecnología del sistema de gestión de base de datos implantada en las instalaciones sobre las que se va a realizar el deployment del sistema a desarrollar, estamos en presencia de una posible concurrencia en la evolución de las fases.

Este fenómeno, en términos de modelización, puede ser asociado a un arreglo unidimensional KT (Knowledge Transferor), compuesto de semáforos KT_f donde $f \in \mathbf{N}$ en el rango [1..n], siendo n la cantidad de fases distinguibles en el C de V y f un identificador de una de sus fases.

El valor de cada semáforo es 0 (cero) si la fase no tiene los *conocimientos* suficientes para continuar y su valor varía de 0 a n dependiendo de la *carga de conocimiento* transferida por la fase *transmisora,* permitiéndole avanzar en la solución. En principio, el vector está inicializado a 0 -al tratarse de un problema desconocido, no hay conocimiento básico para transferir-.

Una abstracción posible sobre este modelo puede expresarse en términos del siguiente *pseudo-código*:

```
/** Declaración e iniciación de un arreglo global de semáforos de transferencia de conocimientos a fases del ciclo de vida **/
var
KT: array [1.. n] of semaphores

Análisis := 1;
Diseño := 2;
.....
Pruebas := n;

Begin
        for i := 1 to n do
                inicia(KT[i], 0)


/** Cómo cada fase es sincronizada dentro de la dinámica del C de V **/
Requisitos : : Fase de C de V
Begin
  while .t. begin
        P(KT [Requisitos]);
                                        /** si tiene conocimientos, o sea, si KT[Requisitos] > 0, avanza,
                                    si no espera por conocimiento transferido desde alguna otra fase **/

        ... evolución de la fase ...

        case conocimientos para transferir a Análisis
                V (KT[Análisis]);
```

/** si tiene conocimientos para transferir a Análisis, ejecuta un V sobre el semáforo de avance de Análisis habilitando a la fase a continuar avanzando en su evolución **/

case conocimientos para transferir a Diseño
V (*KT*[Diseño]);
/** Ídem anterior, pero con respecto a Diseño **/
.....
case conocimientos para transferir a Pruebas
V (*KT*[Pruebas]);

/** Ídem anterior, pero con respecto a Pruebas **/

end.
end.

La sucesión de fases en forma concurrente del Modelo Concurrente de Ciclo de Vida, podría representarse gráficamente tal como se presenta en la siguiente figura:

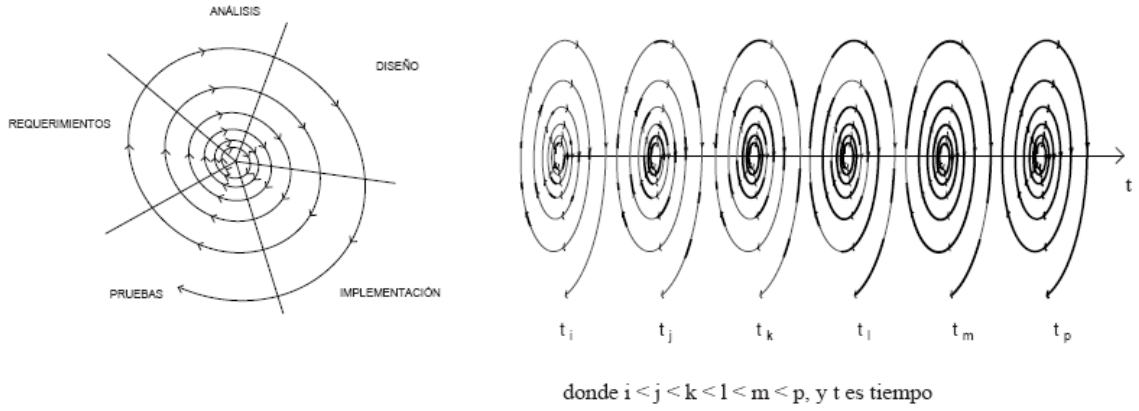

Figura 1 – MCCV

En la gráfica, se ha tomado como modelo de ejemplo, la espiral de Boehm [13]. Si consideramos a su esquema gráfico como el lugar abstracto donde transcurren los eventos de la dinámica asociada al ciclo de vida, se podrán observar las "instantáneas" para cada t_i, t_j, ..., t_p, representada por cada una de las espirales que se suceden en la figura, atravesadas por el eje del tiempo. Se observa, en cada espiral o instante del tiempo, como se va "completando" el ciclo de vida del producto software de manera indeterminística.

Este proceso solo corresponde a ese lapso de tiempo, comprendido entre t_i y t_p, y para un caso en particular. De lo que se desprende que, aunque probabilísticamente posible, a los fines prácticos es imposible encontrar secuencias de evolución iguales.

Dado que el modelo es concurrente y las actividades de cada fase son disparadas de manera no determinística, no se puede predecir cómo se ordenarán las fases del ciclo de vida, lo que revela un nuevo planteo.

- Toda fase existe desde el instante $T_i = 0$;

- Se instancian de manera no determinística, de acuerdo con una función de correspondencia estocástica, con una distribución de probabilidades, generando un posible derrotero del ciclo de vida;
- Sus finalizaciones se desencadenan ordenadamente, de acuerdo con el modelo clásico, a pesar de que el momento de nacimiento de cada fase tiene un grado no determinado de incertidumbre, el orden de finalización está claramente determinado. La probabilidad P de que la fase de requerimientos finalice antes de la fase de implementación es 1, de otro modo, la fase de requerimientos incluiría actividades que no tendrían impacto en la implementación, de lo que se desprendería un cuestionamiento sobre el sentido de estas actividades.

Desde un punto de vista general se puede abordar el tema de la paralelización de las tareas desde dos visiones:

1. La posibilidad llevar a cabo las tareas que deben ser explotadas en paralelo mientras sus dependencias temporales lo permitan.
2. La posibilidad llevar a cabo tareas que se presentan de modo indeterminístico y concurrente.

Estas dos visiones confluyen en un mismo resultado, pero de acuerdo al abordaje que se realice, se planteará la solución.

Incorporando al desarrollo de software los conceptos del Total Quality Management basadas en la detección del proceso de producción ante la presencia de fallos y a través de la mejora continua, suponemos que cada instancia del producto, desarrollada en una fase del MCCV, puede pasar a otra fase del desarrollo, cualquiera que sea, solo si tiene una aceptación de que todo lo necesario para esa parte ha sido completada y si la fase que la requiera para iniciar su ejecución cuenta con los conocimientos necesarios para comenzar dicha instancia, y cada fase continúe trabajando en forma concurrente con otra en la misma parte del producto.

De esta manera, surge la cuestión sobre cuáles son las condiciones para que cada producto en cada fase avance o se detenga. Para ello, el Modelo MCCV incorpora el mecanismo de semáforos, para instanciar los permisos que cada fase tiene para entregar y/o para recibir una parte del producto (conocimiento).

4. Innovaciones Ágiles en el Proceso Software

Los modelos de proceso software que se han desarrollado en los últimos años, conocidos como Métodos Ágiles (MAs) [14] [15] [16], han surgido especialmente en reacción a la complejidad de los modelos tradicionales de desarrollo software y proponen encontrar un equilibrio entre la inexistencia de proceso y un proceso demasiado complejo que en ellas tienen las metodologías y los estándares tradicionales [17].

Para ello proponen la modificación de algunos de los principios que han sido utilizados tradicionalmente por los métodos de desarrollo de software, tales como la producción de gran cantidad de documentación para cada actividad o la definición detallada de tareas y la gran dependencia entre ellas.

En esta diversidad de Métodos, eXtreme Programming (XP) resulta ser el método más utilizado entre los MAs. Por este motivo, en este capítulo nos centraremos en este método.

Las prácticas inicialmente propuestas por XP son las siguientes [18]:

- **Planeamiento de entregas.** Los programadores estiman el esfuerzo necesario para implementar las historias del cliente y éste decide sobre el alcance y la agenda de las entregas. Esta práctica busca determinar rápidamente el alcance de la siguiente versión, combinando prioridades del negocio definidas por el cliente y las estimaciones técnicas de los programadores. Los spikes ("púas" o "astillas") son un experimento dinámico de código y se utilizan para estimar la duración y dificultad de una tarea inmediata. Constituyen la versión ágil de un prototipo.
- **Entregas pequeñas y frecuentes**. Se desarrolla rápidamente un pequeño sistema, al menos uno cada dos o tres meses y se pueden liberar versiones nuevas diariamente a las cuales se agregan pocos elementos cada vez.
- **Prueba continua.** La integración del desarrollo está orientado por las pruebas. El propósito del código no es cumplir un requerimiento, sino pasar las pruebas. Existen dos tipos de prueba: las pruebas unitarias, que verifican una sola clase y las pruebas de aceptación que integran y verifican todo el sistema, o una gran parte. Todos los puntos a desarrollar deben tener test automáticos de unidad y de aceptación para evitar el trabajo de inspección. El cliente ayuda a escribir las pruebas funcionales antes que se escriba el código, si bien las pruebas y el código son escritas por el mismo programador, o par de programadores, las pruebas deben realizarse en forma automática sin intervención humana.
- **Integración continua.** Cada parte que se desarrolla se integra a la base de código apenas está lista. Este proceso puede realizarse varias veces al día, para lo cual, se destina una máquina exclusivamente a este proceso.
- **Todo el equipo en el mismo lugar.** El cliente debe estar presente en el lugar de desarrollo y disponible a tiempo completo para el equipo, para la definición de los requerimientos y la realización de la prueba continua.

Si bien, sus definiciones están lejos de la sistematización del desarrollo, comparten un modelo organizativo incremental, basado en pequeñas entregas con ciclos rápidos, el trabajo debe ser altamente calificado, con trabajo cooperativo donde los desarrolladores y los usuarios trabajan juntos en estrecha comunicación, el método es simple, fácil de aprender y adaptativo en la medida que resulte capaz de incorporar los constantes cambios.

5. Conclusiones y trabajo futuro

En el presente artículo se ha presentado un modelo de ciclo de vida que permite la confluencia de diversos aspectos provenientes de la ingeniería de procesos del modelo toyotista, el MCCV y algunas claves de los postulados agilistas.

La intención del mismo es centrar el análisis en la transformación constante de los elementos organizativos del proceso software, de manera tal que permita introducir en el ciclo de vida del software, el concepto de detención del proceso ante la *presencia de fallos,* la aplicación de *semáforos* en el desarrollo para la detención o continuidad del proceso y la *transferencia de conocimiento*.

Es por ello que se han destacado tres factores diferenciales subtensos en los conceptos principales de los modelos y los postulados presentados.

1. La paralelización y concurrencia de tareas.

2. La detención del proceso, donde la longitud[2] de las líneas de producción, son proporcionales al esfuerzo que conlleva mantener la calidad hasta la confección del entregable final.

3. La incertidumbre sobre el momento en que aparece el "error".

De acuerdo al MCCV, la concurrencia de las posibilidades de progresar en las actividades que provocan los cambios de estado en el ciclo de vida del software se presentan sin ningún tipo de control sobre el flujo temporal de sus apariciones. El Modelo Toyotista incorpora las técnicas de Total Quality Management basadas fundamentalmente en la detención del proceso productivo en cada módulo ante la presencia de fallos.

En la presente investigación y como línea futura de trabajo, queda por definir cuáles son las condiciones para que cada producto en cada fase avance o se detenga. Para ello, el MCCV incorpora el mecanismo de semáforos, para instanciar los permisos que cada fase tiene para entregar y/o para recibir una parte del producto, aplicados también por el toyotismo en la línea de producción. Si bien los conceptos de parar por un error (falta de conocimiento) o avanzar por la solución del error (obtención de conocimiento) resultan los elementos claves, queda aún por determinar las técnicas y/o prácticas que permitirían instanciar el punto de avance o detención en cada una de las fases.

Referencias

[1] *IEEE Standard for Developing Software Life Cycle Processes*, IEEE Standard 1074-1997.

[2] *ISO/IEC International Standard: Information Technology. Software Life Cycle Processes*, ISO/IEC Standard 12207-1995.

[3] *ISO/IEC International Standard: Information Technology. Software Life Cycle Processes*, Amendment 1, ISO/IEC Standard 12207-1995/Amd. 1-2002.

[4] *ISO/IEC 15504, Information Technology – Software Process Assessment*. International Organization for Standardization, International Electrotechnical Commission. 1998. http://isospice.com/standard/tr15504.htm.

[5] Carnegie Mellon, Software Engineering Institute. *Capability Maturity Model® Integration (CMMISM), Version 1.1. CMMI^{SM} for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing. (CMMI-SE/SW/IPPD/SS, V1.1).* Carnegie Mellon University. 2002.

[6] L. Alexander and A. Davis, "Criteria for selecting software process models". *Proceedings of COMPSAC'91*. 521-528. 1991.

[7] Estayno, A. *El Modelo Concurrente del Ciclo de Vida - MCCV - 1205*.(mimeo) 2006.

[8] Coriat. B.: *Pensar al Revés: Trabajo y Organización en la Empresa Japonesa*. Siglo XXI, 1992.

[9] Liker J. *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer*. Mac Graw Hill. 2004

[10] Fruin. M.: *Las Fábricas del Conocimiento, la Administración del Capital Intelectual en Toshiba*. Ed. Oxford, 2000.

[11] E. W. Dijkstra, *Solution of a problem in concurrent programming control.* Communications of ACM. NuevaYork. 1965.

[12] Dijkstra E. W. *Cooperating sequential processes.* University of Texas. NuevaYork. 1968.

[12 bis] A. Siberschatz y P. B. Galvin. *Operating System Concepts*. Addison Wesley. Massachusetts. 1998

[13] B. Boehm. "A Spiral Model of Software Development and Enhancement" *Computer*, pp. 61-78, May 1988.

[2] Usamos aquí la noción de longitud, pero ésta solo trasciende como una metáfora del tiempo que toma completar el camino crítico del cronograma de trabajo.

[14] Beck,K.; Beedle,M.; Cockburn,A.; Cunnimgham,W.; Fowler,M; et al., *Agile Manifesto*. web site. (2001a). http://agilemanifesto.org.
[15] Beck,K.; Beedle,M.; Cockburn,A.; Cunnimgham,W.; Fowler,M; et al., *Principles behind the Agile Manifesto*. web site. (2001b). http://agilemanifesto.org/principles.html.
[16] Fowler, M., Beck,K. Brant, J.; Opdyke W. and Roberts, D. Refactoring: *Improving the design of existing code*. Addison Wesley, 1999.
[17] Acuña, S.; Juristo, N.; Mon, A.; Moreno, A. *A Software Process Model Handbook for Incorporating People's Capabilities.* Springer; 1 Edition, 2005.
[18] Fowler, M. *"Is design dead?" Proceedings XP2000*. Web site . 2001.