Español

Estas declaraciones ejemplifican el debate actual sobre los modelos de proceso del ciclo de vida del software. El tema ha recibido recientemente mucha atención.

El informe del Grupo de Tareas de la Junta de Ciencias de la Defensa sobre el Software Militar, publicado en 1987, puso de relieve la preocupación de que los modelos de proceso de software tradicionales desalentaran enfoques más eficaces para el desarrollo de software, como la creación de prototipos y la reutilización de software. The Computer Society ha patrocinado tutoriales y talleres sobre modelos de procesos de software que han ayudado a aclarar muchos de los problemas y a estimular los avances en el campo (ver "Lectura adicional").

El modelo espiral presentado en este artículo es un candidato para mejorar la situación del modelo de proceso de software. La principal característica distintiva del modelo en espiral es que crea un enfoque orientado al riesgo al proceso de software en lugar de un proceso basado principalmente en documentos o codificado. Incorpora muchas de las fortalezas de otros modelos y resuelve muchas de sus dificultades.

Este artículo se abre con una breve descripción de los modelos de proceso de software y los problemas que abordan. Las secciones subsiguientes describen los pasos del proceso involucrados en el modelo espiral; Ilustran la aplicación del modelo en espiral a un proyecto de software, utilizando como ejemplo el Proyecto de Productividad de Software TRW; Resumir las principales ventajas e implicaciones que implica el uso del modelo espiral y las principales dificultades para utilizarlo en su actual nivel incompleto de elaboración; Y presentar las conclusiones resultantes.

Antecedentes sobre los modelos de proceso de software

Las funciones primarias de un modelo de proceso de software son determinar el orden de las etapas involucradas en el desarrollo y la evolución del software y establecer los criterios de transición para avanzar de una etapa a la siguiente. Estos incluyen criterios de finalización para la etapa actual más criterios de selección y criterios de ingreso para la siguiente etapa. Por lo tanto, un modelo de proceso aborda las siguientes preguntas de proyectos de software:

(1) ¿Qué haremos después?

(2) ¿Cuánto tiempo continuaremos haciéndolo?

En consecuencia, un modelo de proceso difiere de un método de software (a menudo llamado una metodología) en que el enfoque principal de un método es cómo navegar a través de cada fase (determinación de datos, control o '' utiliza jerarquías, funciones de partición, asignación de requisitos) y Cómo representar los productos de la fase (diagramas de la estructura, hilos de la respuesta del estímulo-respuesta, diagramas de la transición del estado).

¿Por qué son importantes los modelos de proceso de software? Principalmente porque proporcionan orientación sobre el orden (fases, incrementos, prototipos, tareas de validación, etc.) en el que un proyecto debe llevar a cabo sus principales tareas. Muchos proyectos de software, como muestra la siguiente sección, han llegado a la pena porque han seguido sus diversas fases de desarrollo y evolución en el orden equivocado.

Evolución de los modelos de procesos. Antes de concentrarnos en profundidad en el modelo espiral, deberíamos echar un vistazo a varios otros: el modelo de código y fijación, el modelo de escenario y el modelo de cascada, el modelo de desarrollo evolutivo y el modelo de transformación.

El modelo de código y corrección. El modelo básico utilizado en los primeros días de software

(1) Escribir un código.

(2) Corregir los problemas en el código.

Así, el orden de los pasos era hacer Algunos de codificación en primer lugar y pensar en los requisitos, diseño, prueba y mantenimiento más adelante. Este modelo tiene tres dificultades principales:

(A) Después de varias correcciones, el código se volvió tan mal estructurado que las correcciones posteriores resultaron muy costosas. Esto subrayó la necesidad de una fase de diseño antes de la codificación.

(B) Frecuentemente, incluso un software bien diseñado era un partido tan pobre para las necesidades de los usuarios que fue rechazado totalmente o se volvió a desarrollar. Esto hizo evidente la necesidad de una fase de requisitos antes del diseño.

(C) El código era costoso de fijar debido a la preparación pobre para la prueba y la modificación. Esto dejó claro que era necesario el reconocimiento explícito de estas fases, así como la planificación de la prueba y la evolución y las tareas de preparación en las primeras fases.

Modelos escenográficos y cascadas. Ya en 1956, la experiencia en grandes sistemas de software como el Semi-Automated Ground Environment (SAGE) había llevado al reconocimiento de estos problemas y al desarrollo de un modelo escénico para abordarlos. Este modelo estipulaba que el software se desarrollara en etapas sucesivas (plan operacional, especificaciones operativas, especificaciones de codificación, codificación, pruebas de parámetros, pruebas de ensamblaje, shakedown, evaluación de sistemas).

El modelo de cascada, 3 ilustrado en la Figura 1, fue un refinamiento de 1970 altamente influyente del modelo escenario. Proporcionó dos mejoras principales al modelo escénico:

1) Reconocimiento de los bucles de retroalimentación entre etapas, y una guía para confinar los bucles de retroalimentación a etapas sucesivas para minimizar la costosa reelaboración involucrada en la retroalimentación a través de muchas etapas.

2) Una incorporación inicial de prototipos en el ciclo de vida del software, a través de un paso de "construirlo dos veces" en paralelo con el análisis de necesidades y el diseño.

El enfoque del modelo de cascada ayudó a eliminar muchas dificultades previamente encontradas en proyectos de software. El modelo de cascada se ha convertido en la base para la mayoría de las normas de adquisición de software en el gobierno y la industria. Algunas de sus dificultades iniciales se han abordado añadiendo extensiones para cubrir desarrollos incrementales, desarrollos paralelos, familias de programas, adaptación de cambios evolutivos, desarrollo y verificación formal de software y validación en etapas y análisis de riesgo.

Sin embargo, incluso con revisiones y mejoras extensas, el esquema básico del modelo de cascada ha encontrado algunas dificultades más fundamentales, y éstas han llevado a la formulación de modelos de proceso alternativos.

Una fuente primaria de dificultad con el modelo de la cascada ha sido su énfasis en documentos completamente elaborados como criterios de terminación para requisitos tempranos y fases de diseño. Para algunas clases de software, como compiladores o sistemas operativos seguros, esta es la forma más eficaz de proceder. Sin embargo, no funciona bien para muchas clases de software, particularmente las aplicaciones interactivas de usuario final. Los estándares basados ​​en documentos han empujado a muchos proyectos a escribir especificaciones elaboradas de interfaces de usuario mal entendidas y funciones de soporte de decisión, seguido por el diseño y desarrollo de grandes cantidades de código inutilizable.

Estos proyectos son ejemplos de cómo los proyectos modelo-cascada han llegado al dolor al perseguir etapas en el orden equivocado. Además, en las áreas soportadas por lenguajes de cuarta generación (hojas de cálculo o aplicaciones de pequeñas empresas), es claramente innecesario escribir especificaciones elaboradas para su aplicación antes de implementarla.

El modelo de desarrollo evolutivo. Las preocupaciones anteriores han conducido a la formulación del modelo de desarrollo evolutivo 4, cuyas etapas consisten en la expansión de los incrementos de un producto de software operacional, y las direcciones de la evolución están determinadas por la experiencia operativa.

El modelo de desarrollo evolutivo está idealmente adaptado a una aplicación de lenguaje de cuarta generación y bien adaptado a situaciones en las que los usuarios dicen: "No puedo decirte lo que quiero, pero lo sé cuando lo veo". Una capacidad operativa inicial rápida y proporciona una base operativa realista para determinar las subsiguientes mejoras del producto.

Sin embargo, el desarrollo evolutivo también tiene sus dificultades. Generalmente es difícil distinguirlo del antiguo modelo de código y fijación, cuyo código de espagueti y falta de planificación fueron la motivación inicial para el modelo de cascada. También se basa en la suposición a menudo poco realista de que el sistema operativo del usuario será lo suficientemente flexible como para acomodarse a caminos de evolución no planificados. Esta suposición no está justificada en tres circunstancias principales:

(1) Circunstancias en las que varias solicitudes desarrolladas de forma independiente deben ser posteriormente estrechamente integradas.

(2) Casos de "esclerosis informacional", en los que los problemas temporales de las deficiencias del software se solidifican cada vez más en obstáculos inalterables a la evolución. El siguiente comentario es un ejemplo típico: "Es bueno que pudieras cambiar esos códigos de equipo para que sean más inteligibles para nosotros, pero el Comité de Códigos acaba de reunirse y establecer los códigos actuales como estándares de la compañía".

(3) Situaciones de puenteo, en las que el nuevo software reemplaza incrementalmente un gran sistema existente. Si el sistema existente está pobremente modularizado, es difícil proporcionar una buena secuencia de "puentes" entre el software antiguo y los incrementos en expansión de un nuevo software.

Bajo tales condiciones, los proyectos de desarrollo evolutivo han llegado a la pena persiguiendo etapas en el orden equivocado: la evolución de una gran cantidad de difícil de cambiar de código antes de abordar la arquitectura de largo alcance y las consideraciones de uso.

El modelo de transformación. Las dificultades con el código de espagueti del desarrollo evolutivo y los modelos de código y fijación también pueden convertirse en una dificultad en varias clases de aplicaciones de modelo de cascada, en las que el código se optimiza para el rendimiento y se vuelve cada vez más difícil de modificar. Se ha propuesto el modelo de transformación como una solución a este dilema.

El modelo de transformación asume la existencia de una capacidad para convertir automáticamente una especificación formal de un producto de software en un programa que satisface la especificación. Los pasos entonces prescritos por el modelo de transformación son

Una especificación formal de la mejor comprensión inicial del producto deseado;

Transformación automática de la especificación en código;

Un bucle iterativo, si es necesario, para mejorar el rendimiento del código resultante dando orientación de optimización al sistema de transformación;

Ejercicio del producto resultante; Y un bucle iterativo externo para ajustar la especificación basada en la experiencia operativa resultante, y para redirigir, reoptimizar y ejercitar el producto de software ajustado.

El modelo de transformación evita así la dificultad de tener que modificar el código que se ha vuelto pobremente estructurado a través de reoptimizaciones repetidas, puesto que las modificaciones se hacen a la especificación. También evita el tiempo extra y el gasto que implica el diseño intermedio, el código y las actividades de prueba.

Sin embargo, el modelo de transformación tiene varias dificultades. Las capacidades de transformación automática sólo están disponibles para pequeños productos en algunas áreas limitadas: hojas de cálculo, pequeñas aplicaciones de lenguaje de cuarta generación y dominios informáticos limitados. El modelo de transformación también comparte algunas de las dificultades del modelo de desarrollo evolutivo, como el supuesto de que los sistemas operativos de los usuarios serán siempre lo suficientemente flexibles como para soportar caminos de evolución no planificados. Además, se enfrentaría a un formidable problema de mantenimiento de la base de conocimientos para hacer frente a la creciente y creciente oferta de componentes de software reutilizables y productos de software comerciales. (Simplemente considere el problema de controlar los costos, el rendimiento y las características de todos los sistemas de administración de bases de datos comerciales y elegir automáticamente el mejor para implementar cada especificación nueva o modificada).

El modelo espiral

El modelo en espiral del proceso de software (ver Figura 2) ha estado evolucionando durante varios años, basado en la experiencia con varios refinamientos del modelo de cascada aplicado a los grandes proyectos de software del gobierno. Como se discutirá, el modelo en espiral puede acomodar la mayoría de los modelos anteriores como casos especiales y proporciona una guía sobre la combinación de modelos anteriores que mejor se adapta a una situación de software dada. El desarrollo del Sistema de Productividad de Software TRW (TRW-SPS), descrito en la siguiente sección, es su aplicación más completa hasta la fecha.

La dimensión radial en la Figura 2 representa el costo acumulativo incurrido en el cumplimiento de los pasos hasta la fecha; La dimensión angular representa el progreso realizado en completar cada ciclo de la espiral. (El modelo refleja el concepto subyacente de que cada ciclo involucra una progresión que aborda la misma secuencia de pasos, para cada porción del producto y para cada uno de sus niveles de elaboración, desde un concepto general de documento de operación hasta la codificación de cada individuo ). Obsérvese que se ha tomado alguna licencia artística con la creciente dimensión de costo acumulativo para mejorar la legibilidad de los pasos de la Figura 2.

Un ciclo típico de la espiral. Cada ciclo de la espiral comienza con la identificación de los objetivos de la porción del producto que se está elaborando (rendimiento, funcionalidad, capacidad para acomodar el cambio, etc.); Los medios alternativos de implementar esta parte del producto (diseño A, diseño B, reutilización, compra, etc.); Y las limitaciones impuestas en la aplicación de las alternativas (costo, horario, interfaz, etc.).

El siguiente paso es evaluar las alternativas relativas a los objetivos y limitaciones. Con frecuencia, este proceso identificará áreas de incertidumbre que son fuentes importantes de riesgo de proyecto. Si es así, el siguiente paso debería incluir la formulación de una estrategia rentable para resolver las fuentes de riesgo. Esto puede implicar la creación de prototipos, la simulación, la evaluación comparativa, la verificación de referencias, la administración de cuestionarios de usuarios, el modelado analítico o combinaciones de éstas y otras técnicas de resolución de riesgos.

Una vez que se evalúan los riesgos, el siguiente paso viene determinado por los riesgos restantes relativos. Si el rendimiento o los riesgos de la interfaz de usuario dominan fuertemente el desarrollo del programa o los riesgos internos de control de interfaces, el siguiente paso puede ser evolutivo: un esfuerzo mínimo para especificar la naturaleza general del producto, un plan para el siguiente nivel de prototipado y El desarrollo de un prototipo más detallado para continuar con la solución de los principales problemas de riesgo.

Si este prototipo es operativamente útil y lo suficientemente robusto como para servir como guía de bajo riesgo en cuanto a qué combinación de modelos anteriores se ajusta mejor a una situación de software dada. El desarrollo del Sistema de Productividad de Software TRW (TRW-SPS), descrito en la siguiente sección, es su aplicación más completa hasta la fecha.

La dimensión radial en la Figura 2 representa el costo acumulativo incurrido en el cumplimiento de los pasos hasta la fecha; La dimensión angu- lar representa el progreso realizado en completar cada ciclo de la espiral. (El modelo refleja el concepto subyacente de que cada ciclo involucra una progresión que aborda la misma secuencia de pasos, para cada porción del producto y para cada uno de sus niveles de elaboración, desde un concepto general de documento de operación hasta la codificación de cada individuo ). Obsérvese que se ha tomado alguna licencia artística con la creciente dimensión de costo acumulativo para mejorar la legibilidad de los pasos de la Figura 2.

Un ciclo típico de la espiral. Cada ciclo de la espiral comienza con la identificación de

Los objetivos de la parte del producto que se está elaborando (desempeño, funcionalidad, capacidad para acomodar el cambio, etc.);

Los medios alternativos de implementar esta parte del producto (diseño A, diseño B, reutilización, compra, etc.); Y las limitaciones impuestas en la aplicación de las alternativas (costo, horario, interfaz, etc.).

El siguiente paso es evaluar las alternativas relativas a los objetivos y limitaciones. Con frecuencia, este proceso identificará áreas de incertidumbre que son fuentes importantes de riesgo de proyecto. Si es así, el siguiente paso debería incluir la formulación de una estrategia rentable para resolver las fuentes de riesgo. Esto puede implicar la creación de prototipos, la simulación, la evaluación comparativa, la verificación de referencias, la administración de cuestionarios de usuarios, el modelado analítico o combinaciones de éstas y otras técnicas de resolución de riesgos.

Una vez que se evalúan los riesgos, el siguiente paso viene determinado por los riesgos restantes relativos. Si los riesgos de desempeño o de interfaz de usuario dominan fuertemente el desarrollo del programa o los riesgos internos de control de interfaces, el siguiente paso puede ser un desarrollo evolutivo: un esfuerzo mínimo para especificar la naturaleza general del producto, un plan para el siguiente nivel de prototipado , Y el desarrollo de un prototipo más detallado para seguir resolviendo los principales problemas de riesgo.

Si este prototipo es operativamente útil y lo suficientemente robusto como para servir como una base de bajo riesgo para la evolución futura del producto, los pasos subsecuentes de riesgo serían la evolución de la serie de prototipos evolutivos hacia la derecha en la Figura 2. En este caso, De las especificaciones de la escritura serían dirigidas pero no ejercitadas. Por lo tanto, las consideraciones de riesgo pueden conducir a un proyecto que implementa sólo un subconjunto de todos los pasos potenciales en el modelo.

Por otro lado, si los esfuerzos previos de prototipado ya han resuelto todos los riesgos de desempeño o de interfaz de usuario y dominan el desarrollo de programas o los riesgos de control de interfaces, el siguiente paso sigue el enfoque básico de cascada (concepto de operación, requisitos de software, diseño preliminar , Etc. en la Figura 2), modificado según sea apropiado para incorporar el desarrollo incremental. Cada nivel de especificación de software en la figura es seguido por un paso de validación y la preparación de planes para el ciclo sucesivo. En este caso, las opciones de prototipo, simular, modelar, etc. son abordadas pero no ejercidas, lo que lleva al uso de un subconjunto de pasos diferente.

Este subconjunto de los pasos del modelo en espiral, impulsado por el riesgo, permite al modelo acomodar cualquier mezcla apropiada de un enfoque orientado a especificaciones, orientado a prototipos, orientado a la simulación, orientado a la transformación automática u otro enfoque para el desarrollo de software. En tales casos, se elige la estrategia mixta apropiada considerando la magnitud relativa de los riesgos del programa y la eficacia relativa de las diversas técnicas para resolver los riesgos. De manera similar, las consideraciones de gestión del riesgo pueden determinar la cantidad de tiempo y esfuerzo que se debe dedicar a otras actividades de proyecto tales como la planificación, la gestión de la configuración, la garantía de calidad, la verificación formal y las pruebas. En particular, las especificaciones basadas en el riesgo (como se analiza en la siguiente sección) pueden tener grados variables de completitud, formalidad y granularidad, dependiendo de los riesgos relativos de hacer muy poca o demasiada especificación.

Una característica importante del modelo en espiral, como con la mayoría de los otros modelos, es que cada ciclo se completa con una revisión que involucra a las personas u organizaciones primarias relacionadas con el producto. Esta revisión abarca todos los productos desarrollados durante el ciclo anterior, incluyendo los planes para el próximo ciclo y los recursos necesarios para llevarlos a cabo. El principal objetivo de la revisión es asegurar que todas las partes interesadas estén mutuamente comprometidas con el enfoque para la siguiente fase.

Los planes para fases posteriores también pueden incluir una partición del producto en incrementos para desarrollo sucesivo o componentes que deben ser desarrollados por organizaciones o personas indicadas. Para este último caso, visualice una serie de ciclos espirales paralelos, uno para cada componente, añadiendo una tercera dimensión al concepto presentado en la Figura 2. Por ejemplo, espirales separadas pueden estar evolucionando para componentes o incrementos de software separados. Por lo tanto, el paso de revisión y omisión puede abarcar desde una revisión del diseño de un componente de un solo programador hasta una revisión de requisitos importantes que involucra a desarrolladores, clientes, usuarios y organizaciones de mantenimiento.

Iniciando y terminando la espiral. Cuatro preguntas fundamentales surgen al considerar esta presentación del modelo espiral:

(1) ¿Cómo comienza la espiral?

(2) ¿Cómo salió de la espiral cuando es apropiado terminar un proyecto temprano?

(3) ¿Por qué la espiral termina tan abruptamente?

(4) ¿Qué sucede con la mejora de software (o mantenimiento)?

La respuesta a estas preguntas implica una observación de que el modelo en espiral se aplica igualmente bien a los esfuerzos de desarrollo o mejora. En cualquier caso, la espiral se inicia con la hipótesis de que una misión operativa (o conjunto de misiones) podría ser mejorada por un esfuerzo de software. El proceso en espiral implica entonces una prueba de esta hipótesis: en cualquier momento, si la hipótesis falla en la prueba (por ejemplo, si los retrasos hacen que un producto de software pierda su ventana de mercado o si un producto comercial superior esté disponible) La espiral se termina. De lo contrario, termina con la instalación de software nuevo o modificado, y la hipótesis se prueba observando el efecto en la misión operativa. Por lo general, la experiencia con la misión operativa conduce a nuevas hipótesis sobre las mejoras de software, y se inicia una nueva espiral de mantenimiento para probar la hipótesis. La iniciación, terminación e iteración de las tareas y productos de los ciclos anteriores se definen implícitamente en el modelo espiral (aunque no se incluyen en la figura 2 para simplificar su presentación).

Utilizando el modelo en espiral

'Las diferentes rondas y actividades involucradas en el modelo espiral se entienden mejor a través del uso de un ejemplo. El modelo de espiral se utilizó en la definición y desarrollo del Sistema de Productividad de Software TRW (TRW-SPS), un entorno de ingeniería de software integrado.6 La oportunidad de misión inicial coincidió con una iniciativa corporativa para mejorar la productividad en todas las operaciones corporativas apropiadas y una hipótesis inicial Que la ingeniería de software era un área atractiva para investigar. Esto condujo a un circuito pequeño, extra "redondo 0" de la espiral para determinar la factibilidad de aumentar la productividad del software a un coste corporativo razonable. (Los proyectos de software muy grandes o complejos preceden frecuentemente a la ronda de "concepto de operación" de la espiral con una o más rondas más pequeñas para establecer viabilidad y reducir la gama de soluciones alternativas rápida y económicamente).

Las Tablas 1, 2 y 3 resumen la aplicación del modelo en espiral a las tres primeras rondas de definición del SPS. Las características principales de cada ronda se discuten posteriormente y son seguidas por algunos ejemplos de rondas posteriores, como el diseño preliminar y detallado.

Ronda 0: Estudio de factibilidad. Este estudio incluyó a cinco participantes a tiempo parcial durante un período de dos a tres meses. Como se indica en la Tabla 1, los objetivos y limitaciones se expresaron a un nivel muy alto y en términos cualitativos como "aumentar significativamente", "a un costo razonable", etc.

Algunas de las alternativas consideradas, sobre todo las del área de "tecnología", podrían conducir al desarrollo de un producto de software, pero el posible atractivo de una serie de alternativas no informáticas en las áreas de gestión, personal e instalaciones podría haber dado lugar a Conclusión de no embarcarse en una actividad de desarrollo de software.

Las principales áreas de riesgo implicaban posibles situaciones en las que la empresa invertiría mucho para descubrir que

- las ganancias de productividad resultantes no fueron significativas, o

- las mejoras potencialmente de alto apalancamiento no eran compatibles con algunos aspectos de la "cultura TRW".

Las actividades de resolución de riesgos emprendidas en la Ronda 0 fueron principalmente encuestas y análisis, incluyendo entrevistas estructuradas con desarrolladores y gestores de software, un análisis inicial de los factores de apalancamiento de la productividad identificados por el modelo de costo constructivo (Cocomo) '; Y un análisis de proyectos anteriores en TRW que exhiben altos niveles de productividad.

Los resultados del análisis de riesgos indicaron que se podrían lograr ganancias de productividad significativas a un costo razonable mediante la adopción de un conjunto integrado de iniciativas en las cuatro esferas principales. Sin embargo, se encontró que algunas soluciones candidatas, tales como un entorno de soporte de software basado en un sistema de tiempo compartido único, corporativo, basado en maxicomputador, estaban en conflicto con restricciones de TRW que requieren soporte de diferentes niveles de proyectos clasificados por seguridad. Así, incluso a un nivel muy alto de generalidad de objetivos y restricciones, la Ronda 0 fue capaz de responder a preguntas básicas de factibilidad y eliminar clases significativas de soluciones candi- datas.

El plan para la primera ronda implicaba un compromiso de 12 meses-hombre en comparación con los dos hombres-meses invertidos en la ronda 0 (durante estas rondas, todos los participantes eran a tiempo parcial). La primera ronda correspondía bastante bien a la ronda inicial del modelo en espiral que se muestra en la Figura 2, ya que su intención era producir un concepto de operación y un plan básico de ciclo de vida para implementar cualquier alternativa preferida que surgiera.

Ronda 1: Concepto de operaciones. La Tabla 2 resume la Ronda 1 de la espiral siguiendo las líneas dadas en la Tabla 1 para la Ronda 0. Las características de la Ronda 1 se comparan con las de la Ronda 0 como sigue:

El nivel de inversión fue mayor (12 frente a 2 hombres / mes).

Los objetivos y limitaciones eran más específicos ("doble productividad de software en cinco años a un costo de $ 1O, OOO una persona" versus "aumentar significativamente la productividad a un costo razonable").

Se presentaron otras limitaciones, como la preferencia por los productos TRW (en particular, un sistema de red de área local (LAN) desarrollado por TRW).

Las alternativas fueron más detalladas ("SREM, PSL / PSA o SADT, como herramientas de requisitos, etc." versus "herramientas", terminales "privados / compartidos", terminales "inteligentes / mudos" versus "estaciones de trabajo").

Las áreas de riesgo identificadas fueron más específicas ("el rendimiento de los precios de LAN de TRW dentro de una restricción de inversión de 510.000 por persona" versus "mejoras pueden violar la restricción de costos razonable").

Las actividades de resolución de riesgos fueron más amplias (incluyendo el benchmark y el análisis de un prototipo de LAN TRW que se está desarrollando para otro proyecto).

El resultado fue un documento de concepto operacional bastante específico, que incluía oficinas privadas adaptadas a patrones de trabajo de software y terminales personales conectados a superminis VAX a través de TRW LAN. Algunas opciones fueron específicamente aplazadas a la siguiente ronda, como la elección del sistema operativo y las herramientas específicas.

El plan de ciclo de vida y el plan para la siguiente fase implicaron una división en actividades separadas para abordar mejoras de gestión, desarrollo de instalaciones y desarrollo del primer incremento de un entorno de desarrollo de software.

El compromiso implicaba más que un simple acuerdo con el plan. Se comprometió a aplicar el entorno a un próximo proyecto de software de prueba de 100 personas y desarrollar un entorno centrado en las necesidades del proyecto de banco de pruebas. También se especificó la formación de un grupo directivo representativo para asegurar que las actividades separadas estuvieran bien coordinadas y que el entorno no fuera demasiado optimizado en torno al proyecto del banco de pruebas.

Aunque el plan recomendó el desarrollo de un entorno prototipo, también recomendó que el proyecto empleara especificaciones de requisitos y especificaciones de diseño de una manera controlada por el riesgo. Así, el desarrollo del ambiente siguió las rondas sucesivas del modelo espiral.

Ronda 2: Especificación de requisitos de nivel superior. La Tabla 3 muestra los pasos correspondientes involucrados durante la Ronda 2 que define el sistema de productividad del software. Las decisiones de la segunda ronda y su racionalidad fueron cubiertas en trabajos anteriores '; Aquí, vamos a resumir las consideraciones relacionadas con la gestión de riesgos y el uso del modelo espiral:

Las actividades iniciales de identificación de riesgos durante la Ronda 2 mostraron que varios requisitos del sistema dependían de la decisión entre un sistema host-target o un conjunto de herramientas totalmente portátil y la decisión entre VMS y Unix como sistema operativo del host. Estos requisitos incluyeron las funciones necesarias para proporcionar un front-end amigable para el usuario, el sistema operativo que utilizarán las estaciones de trabajo y las funciones necesarias para soportar una operación de host-objetivo. Para mantener estos requisitos en sincronización con los demás, se inició un minispiral especial para abordar y resolver estos problemas. La revisión resultante llevó a un compromiso con una operación de host-objetivo utilizando Unix en el sistema host, en un punto lo suficientemente temprano para trabajar los requisitos dependientes del SO de manera oportuna.

Abordar los riesgos de desajustes a las necesidades y prioridades del proyecto del usuario resultó en una participación sustancial del personal del proyecto del usuario en la actividad de definición de requisitos. Esto llevó a varias redirecciones significativas de los requerimientos, particularmente hacia el apoyo a las primeras fases del ciclo de vida del software en el cual el proyecto del usuario se estaba embarcando, como una adaptación de las herramientas de metodología de ingeniería de requisitos de software (SREM) para la especificación y análisis de los requisitos.

También es interesante observar que la forma de las Tablas I, 2 y 3 se desarrolló originalmente con fines de presentación, pero posteriormente se convirtió en una "plantilla de modelo espiral" estándar utilizada en proyectos posteriores. Estas plantillas son útiles no sólo para organizar las actividades del proyecto, sino también como un registro de diseño-justificación residual. La información de justificación del diseño es de suma importancia para evaluar la reutilización potencial de los componentes de software en futuros proyectos. Otro punto importante a destacar es que el uso de la plantilla era uniforme a lo largo de los tres ciclos, mostrando que los pasos en espiral pueden ser y fueron seguidos uniformemente en niveles sucesivamente detallados de definición del producto.

Sucediendo rondas. Sería útil ilustrar algunos ejemplos de cómo el modelo en espiral se utiliza para manejar situaciones que surgen en el diseño preliminar y el diseño detallado de los componentes del SPS: la especificación de diseño preliminar para la herramienta de trazabilidad de los requisitos (RTT) y un diseño detallado O volver a la carpeta de desarrollo de unidades (UDF) herramienta.

La especificación de diseño RTT preliminary. El RTT establece la trazabilidad entre especificaciones detalladas de requisitos de software, elementos de diseño, elementos de código y casos de prueba. También admite varias capacidades asociadas de consulta, análisis y generación de informes. La especificación de diseño preliminar para el RTT (y la mayoría de las otras herramientas SPS) se ve diferente de la especificación de diseño preliminar habitual, que tiende a mostrar un nivel uniforme de elaboración de todos los componentes del diseño. En su lugar, el nivel de detalle de las opciones de generación de informes, una vez que la naturaleza de estas opciones se estableció en algunos ejemplos.

Un diseño detallado de regreso. La herramienta UDF recoge en una "carpeta" electrónica todos los artefactos involucrados en el desarrollo de una unidad de software de un solo programador (normalmente de 500 a 1000 instrucciones): requisitos de unidad, diseño, código, casos de prueba, resultados de pruebas y documentación. También incluye una plantilla de gestión para el seguimiento de la finalización programada y real del programador de cada artefacto.

Una alternativa considerada durante el diseño detallado de la herramienta UDF fue la reutilización de porciones del RTT para proporcionar indicaciones a los requisitos y especificaciones preliminares de diseño de la unidad que se estaba desarrollando. Esto resultó ser una alternativa extremadamente atractiva, no sólo para evitar el desarrollo de software duplicado, sino también para traer a la superficie varios problemas que implican mapeos de muchos a muchos entre requisitos, diseño y código que no se habían considerado en el diseño de la herramienta UDF . Esto llevó a un replanteamiento de los requisitos de la herramienta UDF y del diseño preliminar, lo que evitó una gran cantidad de reelaboración del código que habría sido necesario si el diseño detallado de la herramienta UDF se hubiera procedido de una manera puramente deductiva y descendente de la UDF original especificación de requisitos. El retorno resultante llevó a una herramienta UDF significativamente diferente, menos costosa y más capaz, incorporando el RTT en su "jerarquía de usos".

Características del modelo en espiral. Estos dos ejemplos ilustran varias características del enfoque espiral.

Promueve el desarrollo de especificaciones que no son necesariamente uniformes, exhaustivas u oficiales, ya que aplazan la elaboración detallada de elementos de software de bajo riesgo y evitan la rotura innecesaria en su diseño hasta que se estabilicen los elementos de alto riesgo del diseño.

Incorpora el prototipado como una opción de reducción de riesgos en cualquier etapa del desarrollo. De hecho, los análisis de riesgo de prototipos y reutilización se usaron a menudo en el proceso de pasar del diseño detallado al código.

Se accorpmodates rework o back-backs a etapas anteriores como alternativas más atractivas se identifican o como nuevos problemas de riesgo necesitan resolución.

En general, los documentos basados ​​en el riesgo, en particular las especificaciones y los planes, son características importantes del modelo en espiral. Gran cantidad de detalles no son necesarios a menos que la ausencia de tal detalle pone en peligro el proyecto. En algunos casos, tal como con un producto cuya funcionalidad puede ser determinada por una elección entre productos comerciales, puede ser preferible un conjunto de criterios de evaluación ponderados para los productos a una preinscripción detallada de requisitos funcionales.

**Resultados**. El Sistema de Productividad de Software desarrollado y apoyado utilizando el modelo en espiral evitó los riesgos identificados y logró la mayoría de los objetivos del sistema. El SPS ha crecido hasta incluir más de 300 herramientas y más de 1.300.000 instrucciones; El 93 por ciento de las instrucciones se reutilizaron de los paquetes desarrollados por el proyecto, desarrollados por TRW o externos. Más de 25 proyectos han utilizado todo o partes del sistema. Todos los proyectos que utilizan completamente el sistema han aumentado su productividad al menos 50%; De hecho, la mayoría han duplicado su productividad (cuando se compara con el modelo de estimación de costos predicciones de su productividad utilizando métodos tradicionales).

Sin embargo, una área de riesgo -que los proyectos con sistemas de destino no Unix no aceptarían un sistema host basado en Unix- fue subestimada. Algunos proyectos aceptaron el enfoque de host-target, pero por varias razones (como restricciones de clientes y máquinas de destino de costo cero) muchos no lo hicieron. Como resultado, el sistema fue menos utilizado en los proyectos de TRW de lo esperado. Esta y otras lecciones aprendidas han sido incorporadas al enfoque del modelo en espiral para desarrollar el entorno de desarrollo de software de próxima generación de TRW.

Evaluación

Ventajas. La principal ventaja del modelo espiral es que su gama de opciones satisface las buenas características de los modelos de proceso de software existentes, mientras que su enfoque basado en el riesgo evita muchas de sus dificultades. En situaciones apropiadas, el modelo en espiral se convierte en equivalente a uno de los modelos de proceso existentes. En otras situaciones, proporciona orientación sobre la mejor combinación de enfoques existentes para un proyecto dado; Por ejemplo, su aplicación al TRW-SPS proporcionó una combinación de especificación, creación de prototipos y desarrollo evolutivo basada en el riesgo.

Las condiciones principales bajo las cuales el modelo espiral se convierte en equivalente a otros modelos de proceso principales se resumen de la siguiente manera:

Si un proyecto tiene un riesgo bajo en este proyecto de áreas. En algunos casos, tal como con un producto cuya funcionalidad puede ser determinada por una elección entre productos comerciales, puede ser preferible un conjunto de criterios de evaluación ponderados para los productos a una preinscripción detallada de requisitos funcionales.

Resultados. El Sistema de productividad de software desarrollado y apoyado utilizando el modelo en espiral evitó los riesgos identificados y logró la mayoría de los objetivos del sistema. El SPS ha crecido hasta incluir más de 300 herramientas y más de 1.300.000 instrucciones; El 93 por ciento de las instrucciones se reutilizaron de paquetes desarrollados por el proyecto anterior, desarrollados por TRW o externos. Más de 25 proyectos han utilizado todo o partes del sistema. Todos los proyectos que utilizan completamente el sistema han aumentado su productividad al menos 50%; De hecho, la mayoría han duplicado su productividad (cuando se compara con el modelo de estimación de costos predicciones de su productividad utilizando métodos tradicionales).

Sin embargo, una área de riesgo -que los proyectos con sistemas de destino no Unix no aceptarían un sistema host basado en Unix- fue subestimada. Algunos proyectos aceptaron el enfoque de host-target, pero por varias razones (como restricciones de clientes y máquinas de destino de costo cero) muchos no lo hicieron. Como resultado, el sistema fue menos utilizado en los proyectos de TRW de lo esperado. Esta y otras lecciones aprendidas han sido incorporadas al enfoque del modelo en espiral para desarrollar el entorno de desarrollo de software de próxima generación de TRW.

Evaluación

Ventajas. La principal ventaja del modelo espiral es que su gama de opciones satisface las buenas características de los modelos de proceso de software existentes, mientras que su enfoque basado en el riesgo evita muchas de sus dificultades. En situaciones apropiadas, el modelo en espiral se convierte en equivalente a uno de los modelos de proceso existentes. En otras situaciones, proporciona orientación sobre la mejor combinación de enfoques existentes para un proyecto dado; Por ejemplo, su aplicación a la TRW-SPS proporcionó una mezcla basada en el riesgo de especificación, creación de prototipos y desarrollo evolutivo.

Las condiciones principales bajo las cuales el modelo espiral se convierte en equivalente a otros modelos de proceso principales se resumen de la siguiente manera:

Si un proyecto tiene un bajo riesgo en áreas tales como obtener la interfaz de usuario equivocada o no cumplir con los rigurosos requisitos de desempeño y si tiene un alto riesgo de previsibilidad y control del presupuesto y del calendario, estas consideraciones de riesgo llevan al modelo espiral a una equivalencia a El modelo de la cascada.

Si los requisitos de un producto de software son muy estables (lo que implica un bajo riesgo de costoso diseño y rotura de código debido a cambios de requisitos durante el desarrollo), y si la presencia de errores en el producto de software constituye un alto riesgo para la misión a la que sirve, Las consideraciones hacen que el modelo espiral se asemeje al modelo de dos piernas de especificación precisa y desarrollo de programa deductivo formal.

Si un proyecto tiene un bajo riesgo en áreas tales como la pérdida de presupuesto y previsibilidad y control de horarios, problemas de integración en grandes sistemas o enfrentamientos con la esclerosis de información y si tiene un alto riesgo en áreas tales como obtener la interfaz de usuario o usuario errónea Los requisitos de apoyo de decisión, entonces estas consideraciones de riesgo impulsar el modelo espiral en una equivalencia con el modelo de desarrollo evolutivo.

Si las capacidades de generación automatizada de software están disponibles, el modelo en espiral las acomoda como opciones para el prototipado rápido o para la aplicación del modelo de transformación, dependiendo de las consideraciones de riesgo involucradas.

Si los elementos de alto riesgo de un proyecto implican una mezcla de los elementos de riesgo enumerados anteriormente, entonces el enfoque en espiral reflejará una mezcla apropiada de los modelos de proceso anteriores (como se ejemplifica en la aplicación de TRW-SPS). Al hacerlo, sus características de evitación del riesgo generalmente evitarán las dificultades de los otros modelos.

El modelo espiral tiene una serie de ventajas adicionales, resumidas de la siguiente manera:

Enfoca la atención temprana en las opciones que implican la reutilización del software existente.

Las etapas que involucran la identificación y evaluación de alternativas alientan estas opciones.

Proporciona una preparación para la evolución del ciclo de vida, el crecimiento y los cambios del producto lógico.

Las principales fuentes de cambio de producto están incluidas en los objetivos del producto, y los enfoques de información son alternativas atractivas de diseño arquitectónico en que reducen el riesgo de no poder acomodar los objetivos de la carga del producto.

Proporciona un mecanismo para incorporar objetivos de calidad de software en el desarrollo de productos de software. Este mecanismo se deriva del énfasis en identificar todos los tipos de objetivos y limitaciones durante cada ronda de la espiral. Por ejemplo, la Tabla 3 muestra la facilidad de uso, la portabilidad y la fiabilidad como objetivos y restricciones específicos a ser abordados por el SPS. En la Tabla I, las restricciones de seguridad se identificaron como un elemento de riesgo clave para el MSF.

Se centra en la eliminación temprana de errores y alternativas poco atractivas. Las medidas de análisis de riesgos, validación y compromiso abarcan estas consideraciones.

Para cada una de las fuentes de la actividad del proyecto y el gasto de resource, responde a la pregunta clave, "¿Cuánto es suficiente?" De otra manera, "¿Cuánto de los requisitos de análisis, planificación, gestión de la configuración, garantía de calidad, . "Si se utiliza un enfoque basado en el riesgo, se puede ver que la respuesta no es la misma para todos los proyectos y que el nivel de esfuerzo adecuado está determinado por el nivel de riesgo que supone no hacer lo suficiente.

No implica enfoques separados para el desarrollo de software y mejora de software (o mantenimiento). Este aspecto ayuda a evitar el estatus de "ciudadano de segunda clase" frecuentemente asociado con el mantenimiento del software. También ayuda a evitar muchos de los problemas que actualmente se producen cuando los esfuerzos de mejora de alto riesgo se abordan de la misma manera que los esfuerzos de mantenimiento de rutina.

Proporciona un marco viable para el desarrollo de sistemas integrados de hardware y software. El enfoque en la gestión de riesgos y en la eliminación de alternativas poco atractivas temprana y barata es igualmente aplicable al hardware y software.

Dificultades. El modelo espiral completo se puede aplicar con éxito en muchas situaciones, pero algunas dificultades deben ser abordadas antes de que pueda llamarse un modelo maduro, universalmente aplicable. Los tres desafíos principales consisten en hacer coincidir con el software de contratación, basándose en la experiencia en evaluación de riesgos y en la necesidad de una mayor elaboración de los pasos del modelo en espiral.

Emparejar para contratar el software. El modelo en espiral funciona actualmente bien en desarrollos de software internos como el TRW-SPS, pero necesita más trabajo para adaptarlo al mundo de la adquisición de software de contrato. Los desarrollos internos de software tienen una gran flexibilidad y libertad para acomodar los compromisos de cada etapa , Aplazar compromisos con opciones específicas, establecer minispirales para resolver temas críticos, ajustar niveles de esfuerzo o acomodar prácticas tales como prototipos, desarrollo evolutivo o diseño a costo. El mundo de la adquisición de software contractual tiene más dificultades para alcanzar estos grados de flexibilidad y libertad sin perder la responsabilidad y el control, y un tiempo más difícil definir los contratos cuyos entregables no están bien especificados por adelantado.

Recientemente, se han logrado muchos avances en el establecimiento de mecanismos contractuales más flexibles, como el uso de contratos front-end competitivos para la definición de conceptos o prototipos de despegues, el uso de contratos de nivel de esfuerzo y de adjudicación para El desarrollo evolutivo y el uso de contratos de diseño a costo. Aunque estos han sido generalmente exitosos, los procedimientos para usarlos aún necesitan ser resueltos hasta el punto de que los gerentes de adquisición se sienten completamente cómodos al usarlos.

Basándose en la experiencia en evaluación de riesgos. El modelo en espiral depende en gran medida de la capacidad de los desarrolladores de software para identificar y gestionar las fuentes de riesgo del proyecto.

Un buen ejemplo de esto es la especificación conducida por el riesgo del modelo espiral, que lleva los elementos de alto riesgo hasta un gran detalle y deja elementos de bajo riesgo a ser elaborados en etapas posteriores; En este momento, hay menos riesgo de rotura.

Sin embargo, un equipo de desarrolladores inexpertos o de bajo nivel puede también producir una especificación con un patrón diferente de variación en los niveles de detalle: una gran elaboración de detalle para los elementos bien entendidos y de bajo riesgo, y poca elaboración de lo mal entendido , Elementos de alto riesgo. A menos que haya una revisión perspicaz de tal especificación por el personal experimentado del desarrollo o de la adquisición, este tipo de proyecto dará una ilusión del progreso durante un período en el cual está dirigiendo realmente al desastre.

Otra preocupación es que una especificación basada en el riesgo también dependerá de las personas. Por ejemplo, un diseño producido por un experto puede ser implementado por no expertos. En este caso, el experto, que no necesita una gran cantidad de documentación detallada, debe producir suficiente documentación adicional para evitar que los no expertos se desvien. Los revisores de la especificación también deben ser sensibles a estas preocupaciones.

Con un enfoque convencional basado en documentos, el requisito de llevar todos los aspectos de la especificación a un nivel uniforme de detalle elimina algunos problemas potenciales y permite una revisión adecuada de algunos aspectos por revisores sin experiencia. Pero también crea un drenaje grande en el tiempo de los expertos escasos, que deben cavar para las ediciones críticas dentro de una gran masa de detalle no crítico. Además, si los elementos de alto riesgo han sido ignorados por referencias impresionantes a las capacidades poco entendidas (como un nuevo concepto de sincronización o un SGBD comercial), existe un riesgo aún mayor de que el enfoque convencional dé la ilusión de progreso En situaciones que se dirigen realmente hacia el desastre.

Necesidad de una mayor elaboración de modelos en espiral. En general, los pasos del proceso del modelo en espiral necesitan una mayor elaboración para asegurar que todos los participantes en el desarrollo de software estén operando en un contexto consistente.

Algunos ejemplos de esto son la necesidad de definiciones más detalladas de la naturaleza de las especificaciones e hitos del modelo en espiral, la naturaleza y objetivos de las revisiones del modelo en espiral, las técnicas para estimar y sincronizar los calendarios y la naturaleza de los indicadores de estado del modelo en espiral, Procedimientos de seguimiento del progreso. Otra necesidad es que las directrices y listas de verificación identifiquen las fuentes más probables de riesgo del proyecto y las técnicas de resolución de riesgos más eficaces para cada fuente de riesgo.

Las personas altamente experimentadas pueden utilizar con éxito el enfoque en espiral sin estas elaboraciones. Sin embargo, para un uso a gran escala en situaciones en las que las personas aportan bases de experiencias muy diferentes al proyecto, son importantes los niveles de elaboración adicionales -que se han acumulado a lo largo de los años para los enfoques basados ​​en documentos- para asegurar una interpretación y un uso consistentes de la espiral Enfoque en todo el proyecto.

Los esfuerzos para aplicar y refinar el modelo en espiral se han centrado en la creación de una disciplina de gestión de riesgos de software, incluyendo técnicas para la identificación de riesgos, análisis de riesgos, priorización de riesgos, planificación de gestión de riesgos y seguimiento de elementos de riesgo. La lista de los diez elementos prioritarios de riesgo de software que se dan en la Tabla 4 es un resultado de esta actividad. Otro ejemplo es el plan de gestión de riesgos discutido en la siguiente sección.

Implicaciones: El Plan de Gestión de Riesgos. Incluso si una organización no está preparada para adoptar el enfoque espiral completo, una técnica característica que se puede adaptar fácilmente a cualquier modelo de ciclo de vida proporciona muchos de los beneficios del enfoque en espiral. Este plan, básicamente, asegura que cada proyecto haga una identificación temprana de sus ítems de mayor riesgo (el número 10 no es un requisito absoluto), desarrolla una estrategia para resolver los ítems de riesgo, identifica y establece Una agenda para resolver nuevos temas de riesgo a medida que surgen, y destaca el progreso en comparación con los planes en las revisiones mensuales.

El Plan de Gestión de Riesgos ha sido utilizado con éxito en TRW y otras organizaciones. Su uso ha asegurado un enfoque apropiado en el prototipado temprano, la simulación, el benchmarking, las medidas de personal de persona clave y otras técnicas de resolución temprana del riesgo que han ayudado a evitar muchos proyectos potenciales de "show-stoppers". Software de gestión, DoD-Std-2167, requiere que los desarrolladores de producir y utilizar los planes de gestión de riesgos, al igual que su contraparte US Air Force regulación, AFR 800-14.

En general, el Plan de Gestión de Riesgos y el conjunto de técnicas de maduración de la gestión del riesgo de software proporcionan una base para adaptar los conceptos de los modelos en espiral a los procedimientos de adquisición y desarrollo de software más establecidos.

Podemos extraer cuatro conclusiones a partir de los datos presentados:

(1) el modelo espiral es más adaptable a la gama completa de situaciones de proyectos de software que los enfoques principalmente basados ​​en documentos como el modelo de cascada o los enfoques principalmente codificados como el desarrollo evolutivo. Es particularmente aplicable a sistemas de software muy grandes, complejos y ambiciosos.

(2) El modelo espiral ha sido bastante exitoso en su mayor aplicación hasta la fecha: el desarrollo y mejora de la TRW-SPS. En general, logró un alto nivel de capacidad de soporte de soporte lógico en un tiempo muy corto y proporcionó la flexibilidad necesaria para acomodar un rango dinámico alto de alternativas técnicas y objetivos de usuario.

(3) El modelo espiral todavía no está tan completamente elaborado como los modelos más establecidos. Por lo tanto, el modelo en espiral puede ser aplicado por personal experimentado, pero necesita mayor elaboración en áreas tales como contratación, especificaciones, hitos, revisiones, programación, monitoreo de estado y identificación de área de riesgo para ser completamente utilizable en todas las situaciones.

(4) Las implementaciones parciales del modelo espiral, como el Plan de Gestión de Riesgos, son compatibles con la mayoría de los modelos de proceso actuales y son muy útiles para superar las principales fuentes de riesgo del proyecto.

Expresiones de gratitud

Quiero agradecer a Frank Belz, a Lolo Penedo, a George Spadaro, a Bob Williams, a Bob Balzer, a Gillian Frewin, a Peter Hamer, a Manny Lehman, a Lee Osterweil, a Dave Parnas, a Bill Riddle, Steve Squires ya Dick Thayer, Los revisores informáticos de este artículo, por sus estimulantes y perspicaces comentarios y discusiones de versiones anteriores del artículo, y Nancy Donato por producir sus varias versiones.