UML: El giro positivo

Por Bertrand Meyer

Versión original (en inglés): http://archive.eiffel.com/doc/manuals/technology/bmarticles/uml/page.html.
Traducción al español: Javier Smaldone (17/11/2006).

De: Cándido Smith, alumno de OO-101
A: Profesor Severa Stern
Asunto: Mi solicitud para el cambio de calificación

Estimado Profesor Stern:

El auxiliar de cátedra de su clase OO-101 me ha indicado que le escriba directamente sobre la calificación D-menos que obtuve en mi artículo "*Una evaluación sobre el Lenguaje de Modelado Unificado (UML) propuesto*". Espero que usted considere cambiarla por una mejor (¿quizás una D?), ya que sería un alivio para mi promedio de calificaciones, ya debilitado luego del "Reprobado" que me puso en su última clase. (Quizás recuerde que en el examen final escribí "*debe haber otras cosas entre las rebanadas de pan y Java*". Ahora me doy cuenta de lo poco aconsejable que fue ese comentario, y sinceramente pido disculpas si herí los sentimientos de alguien.)

Por supuesto, me doy cuenta del motivo de mi D-menos y aprecio su generosidad en no ser haber sido más duro. Como el auxiliar de cátedra me indicó, ¡no hay nada positivo sobre UML en mi artículo! Seguramente no puede ser correcto. Todo el mundo sabe que UML abre una brecha en la ingeniería de software, ¿y quién soy yo para cuestionar esto? Este es el por qué no le pido que cambie mi nota sólo por el efecto en mi promedio, aunque espero entenderá que no es agradable perder mi seguro de Buen Estudiante, por no mencionar novias y cosas por el estilo. No, admito que estaba equivocado y quiero enmendar mi error. Debe haber algo bueno que decir sobre UML.

Y puedo asegurarle que aprenderé la lección: ser positivo. Cualquiera sea el tema, siempre es posible darle un giro positivo. ¡El diario de esta mañana hasta imprimió un adjetivo que puede ser interpretado como no poco favorecedor en un artículo sobre Newt Ginrich! ¿Por qué entonces no sobre UML?

¡Se positivo!

Por lo tanto he seguido el consejo del auxiliar de cátedra. Por ejemplo, podría haber cosas buenas que decir sobre la notación misma. Podría ser simple, usable, convincente, fácil de aprender. Y hay de hecho tal notación para el análisis y el diseño orientados a objetos, cuyo conjunto completo de símbolos gráficos cabe en una página y cubre todas las técnicas básicas de descripción de sistemas Orientadas a Objetos, y el cual es particularmente bueno al escalar hasta la descripción de sistemas de gran envergadura: la Notación de Objetos de Negocios (BON) de Waldén y Nerson, como está descripta en su libro [3]. Por supuesto UML no tiene ninguna de esas propiedades. En su intento de mostrar que ha incluido las ideas de todos, es un desborde de símbolo tras símbolo bizarros. Solo el "**Resumen de la notación**" ocupa 60 páginas y ¡tiene su propio índice! UML es, de hecho, tan complejo como un lenguaje de programación grande y críptico, con un uso generoso de "**$**" y "**#**" y "**-**" y "*****" y "**triángulos sólidos sin cola**" y rectángulos y

diamantes y líneas sólidas y líneas punteadas y elipses sólidas y elipses punteadas y flechas de todo tipo y palabras reservadas tomo "**const**" y "**sorted**" (no confundir con "**ordered**") y semánticas distintas para una clase dependiendo si su nombre aparece en letra "romana" o "itálica". ¡Pero al menos un lenguaje de programación, incluso el peor de ellos, es ejecutable! Aquí hay que aprender toda esa complejidad monstruosa para construir diagramas de un posible sistema futuro.

Lo cual nos lleva a la pregunta del desarrollo continuo ("seamless"). Una vez que usted tiene su hermoso (o no tan hermoso) diagrama, querrá construir un sistema, salvo que el presupuesto ya se haya gastado en herramientas CASE (algo común para empresas que toman demasiado enserio la publicidad exagerada de la "metodología" y terminan sin dinero para el desarrollo real). Pero entonces debe recomenzar con un lenguaje de programación para hacer el trabajo real. ¿Y cómo mantiene la consistencia entre el programa y los diagramas? Waldén y Nerson convincentemente discuten el objetivo de continuidad (proveer un proceso único y continuo) y reversibilidad (ser capaces de moverse hacia atrás y hacia adelante entre el análisis, el diseño y la implementación). Herramientas como EiffelBench y EiffelCase soportan esta inquietud, pero no parece ser para nada una inquietud con UML. Pero por supuesto cualquiera que use UML debe ser listo -no solamente para aprender todos los símbolos- y por lo tanto hará un análisis correcto desde la primera vez, luego un diseño correcto desde la primera vez y luego nunca tendrá que cambiar nada en los próximos diez años de la vida del proyecto. O quizás UML es para proyectos cuyas especificaciones nunca cambian. Mi abuela me contó que una vez escuchó sobre un proyecto así en su juventud. Creo que dijo que tenía algo que ver con calcular el 6to. número de Fibonacci.

¿Orientado a Objetos?

Así es que tengo que buscar en otra parte para encontrar algo positivo que decir sobre UML. Estoy tratando con esfuerzo, y espero que lo tome en cuenta antes de enviar las notas finales a la administración. (Recuerde, sólo estoy pidiendo una D, aunque por supuesto una C-menos sería más que apreciada.) Por ejemplo, he intentado ver si podría caracterizar a UML como "orientado a objetos"; todos sabemos que este es un gran logro. Gran oportunidad. Por supuesto los autores hacen uso de los requeridos "**objeto**", "**herencia**", etc. Pero una simple mirada a los diagramas muestra a UML como lo que es: una extensión del modelado de entidad-relación. Los ejemplos básicos muestran asociaciones binarias y ternarias, tales como (página 16 de [1]) asociaciones entre "**vuelo**", "**asiento**" y "**persona**"; esto es diametralmente opuesto al diseño orientado a objetos, donde, como todos sabemos, el mundo está estructurado en clases, construido sobre tipos de objetos y cada operación o propiedad pertenece a una clase. Seguramente en diseño orientado a objetos, ¡no podrá tener un enlace "**pasajero**" que trate simétricamente a "**asiento**", "**vuelo**" y "**persona**"! En un sistema orientado a objetos, pertenecerá a una de las clases; así es como se obtiene la consistencia, simplicidad, modularidad y reusabilidad de la arquitectura OO. Fíjese en BON o Eiffel y disfrute los resultados. Los autores de UML saben esto, por supuesto; para entender por qué llaman a UML orientado a objetos debemos apreciar su famoso sentido del humor. Obviamente lo dicen en broma. (¿Es esto lo suficientemente positivo?)

La evidencia posterior de la broma es provista por la referencia frecuente a los "casos de uso" como un elemento central del método. Cuando "caso de uso" era la consigna de la Web, traté de entender de qué se trataba, y me fue difícil hasta que le pregunté a mi abuelo, quien me lo explicó todo: es el nuevo nombre para el análisis funcional descendente (top-down) de su adolescencia. Uno se fija en qué debe hacer el sistema ("casos de uso") y deduce su arquitectura de ese análisis. Esto es diametralmente opuesto al diseño orientado a objetos, el cual conscientemente rechaza prestar demasiada atención, durante las primeras

etapas, a las funciones principales del sistema, porque están muy sujetas a cambios, porque reproducen el comportamiento de sistemas previos (aquellos que estamos tratando de reemplazar con el nuevo sistema), porque llevan a un compromiso temprano con el orden de operación (¿el aspecto más volátil del software?) y porque se enfocan en las cualidades superficiales del sistema -su interfaz con el resto del mundo- en vez de en sus propiedades fundamentales. En cambio, el diseño OO se concentra en el tipo de los objetos manipulados por el sistema, y define cada uno de esos tipos a través de la lista de todas las operaciones aplicables y sus propiedades abstractas -contratos- sin importar el orden de aplicación. Mi abuelo agregó que estaba agradecido de que los casos de uso estuvieran presentes en UML, porque le traían recuerdos de los buenos viejos tiempos y que nunca le gustaron los objetos. (Creo que este *es* un comentario positivo.)

Encontrando un uso para UML

Obviamente UML no será útil a ningún desarrollador de software. ¿Podría quizás ser útil a alguien más? ¿Líderes de proyectos, quizás? ¿Ejecutivos de empresas? ¿Las empresas mismas? Por supuesto, no. De hecho, la documentación no hace ninguna pretensión de tal utilidad, y por buenas razones. Es difícil de imaginar qué benericio podría obtener un negocio de páginas de diagramas crípticos sobre propiedades confusas de un sistema pobremente comprendido.

Por lo tanto, busco más allá. A veces una propuesta ofrece una solución fallida, pero tiene el mérito de plantear el problema correcto. ¿Podemos decir esto de UML? Espero que sí. Aunque sea podría ayudar a mi promedio y, haciendo esto, me ayudaría a levantar mi autoestima, por no mencionar las novias y cosas por el estilo, pero me estoy desviando. Dicho sea de paso, ¿a qué problema apunta UML? He tratado con empeño, en los interminables documentos del sitio web de Rational, de encontrar una descripción del problema (lo cual se nos enseñó en nuestros cursos de ingeniería, debería aparecer en la introducción de cualquier documento técnico). Me temo que no he encontrado nada útil que reportar.

Amanda al rescate

Para ver qué metas debería haber perseguido, revisé el artículo de mi amiga Amanda Suertuda, la única que obtuvo un A-mas: "*Los desafíos de la industria del software*". (De paso, profesor, no logro comprender por qué Amanda siempre consigue los temas interesantes.) Su reporte, muy agradable debo confesar, hace un buen trabajo al describir los obstáculos técnicos que los desarrolladores deben superar. El buen software debe ser correcto; olvidemos una aserción en una rutina, y tendremos un desastre de u$s500 millones como la reciente explosión del Ariane 5, resultado de un intento equivocado de reutilizar una rutina de Ada sin un mecanismo de aserciones como el de Eiffel (ver la edición de enero de 1997 de IEEE Computer). No veo nada en UML que ayude a la corrección; sólo algo de palabrería dedicada a la noción de contrato, pero los autores muestran que no entienden nada de la idea del Diseño por Contratos (por ejemplo, ¡mezclan requerimientos semánticos con simples declaraciones de tipos!). El buen software debe ser robusto. ¿Cómo ayuda UML? El buen software debe ser fácil de modificar (Amanda llama a esto "*extensibilidad*"); pero el aparato pesado de UML garantiza cualquier cosa menos que los desarrolladores serán capaces de producir alguna descripción del sistema en la cual no será horroroso cambiar algo. El buen software debe ser reutilizable; nada en UML apunta a los desafíos de la reutilización, tales como las convenciones de estandarización de interfaces, separación de comandos y opciones, separación de comandos y consultas, etcétera. El buen software debe ser eficiente -oh, lo lamento, esto está relacionado a la implementación, y no hablamos sobre implementación en compañía educada. (Si UML se ocupara de la

implementación, debería ocuparse de cuestiones relacionadas con el software; lo bueno de las burbujas y flechas, en oposición a los programas, es que nunca se cuelgan.)

En cambio, los documentos de UML parecen tener un único objetivo: una y otra vez convencer al lector de que no ha omitido ninguna consigna, como en este extracto, página 31 de [2], que intenta explicar que los patrones (patterns) son interesantes pero están cubiertos por el concepto de los autores de "*diagrama de interacción*", cualquier cosa que esto sea (no puedo imaginarlo):

> El aspecto interesante de muchos patrones es su comportamiento dinámico. Ciertamente podemos mostrar esto con diagramas de interacción, pero sólo en el contexto de un modelo específico. Es difícil mantener la "patronidad" de los patrones abierta. A fin de cuentas, los patrones son plantillas (en algún sentido), en los cuales las clases, asociaciones, atributos y operaciones pueden estar asignadas a distintos nombres, manteniendo la esencia del patrón. Necesitamos una forma de parametrizar claramente los patrones. Sin embargo, creemos que tenemos suficientes formas de capturar patrones en UML expresándolos en términos de interacciones.

Francamente, ¿quién está interesado en esta farsa? ¿Cómo puede alguien creer que tiene algo que ver (en algún sentido) con la industria del software? Y ni siquiera he citado cosas sobre el "**metamodelo**". Quizás usted podría pedirle a Amanda que dedique su próximo artículo a la "**patronidad**". Hablando de desafíos.

Un amigo que asistió recientemente a una conferencia sobre orientación a objetos me contó sobre las bromas sobre UML que los expertos en OO -algunas de las personas más respetadas en el área- intercambiaron durante los intervalos y en el fondo de las salas. Él no podía creer el contraste entre el bombo público y el menosprecio privado. Pero los CEOs y otros tomadores de decisiones sólo escuchan el bombo publicitario; se les dice que UML es orientado a objetos o, más a menudo, que orientación a objetos significa UML. Cuando no pueda ayudar al desarrollo de software, UML le dará un mal nombre a todo el campo de la OO. Dado los bombardeos de mercadotecnia a su alrededor, UML, al no cumplir con sus promesas, tiene el potencial de retrasar el progreso de la tecnología de objetos por diez años.

¿La respuesta al final?

Así pues, profesor, ¿qué puedo decir, qué puedo decir? Fui con el auxiliar de cátedra y le pregunté si decir "*La página principal de Rational tiene lindos colores*" ayudaría. Pero me dijo que no, que tendría que encontrar algo más sustancioso. Intenté con "*al menos en su última revisión han dejado de llamar a sus cosas "método", lo que muestra que tienen algún sentido del ridículo*". Tampoco le bastó. Así que busqué y busqué y, finalmente -estoy tan excitado- ¡lo encontré! ¡Sí, UML tiene un propósito después de todo!

La razón por la cual no lo descubrí antes es que sólo aparece en la conclusión. Un lugar extraño para describir sobre qué uno estuvo escribiendo, pero mejor que no hacerlo en ninguna parte. Por supuesto que lo que encontré no es un objetivo técnico (UML, como ya dije, no sirve a ningún propósito relacionado con el software, lo cual está bien para mí -algunas personas tienen mejores cosas que hacer con sus vidas que tratar de mejorar la tecnología del software). No es un objetivo gerencial, al menos no para personas que administren proyectos de software. No es un objetivo de negocios, al menos no para negocios que usen UML. Pero es un objetivo.

Cuando finalmente descubrí el objetivo, casi rompo en llanto por el espíritu generoso y noble de los autores. Sería injusto decir que las parvas de documentación de UML están vacías de toda substancia, cuando de hecho contienen una idea genuina. La encontré en el último párrafo del último reporte que describe la revisión 0.91 [2]. Allí estaba, con toda franqueza, explicando todo lo que había malinterpretado en mi joven inocencia. ¿Cómo podría criticar el método por no ayudar a los desarrolladores de software o gerentes, cuando no se ocupa en absoluto del desarrollo de software, sino sólo de desarrollar un mercado para consultores y capacitadores? Todo comenzó a tener sentido: la complejidad y rareza de la notación, lo que tontamente tomé como una deficiencia, son de hecho unas de sus cualidades más atractivas, dado que crean infinitas oportunidades de negocios, para Rational y quizás también para otros; así como también su tibia adopción de las ideas de la orientación a objetos, lo que significa que un consultor no tiene que conocer ni apreciar la tecnología de objetos para sacar provecho de UML.

Mi larga búsqueda no ha sido en vano. Me ha llevado a una total comprensión de UML, su admirable máquina auto-alimentada, dedicada de la A a la Z a la creación de un nuevo mercado, libre de todas las dificultades asociadas al desagradable negocio del desarrollo de software: ¡Libros de UML! ¡Cursos de UML! ¡Cursos sobre los libros! ¡Libros sobre los cursos! ¡Revisiones! ¡Journals de UML! ¡Conferencias! ¡Workshops! ¡Tutoriales! ¡Estándares! ¡Comités! ¡Remeras! Mientras más se piensa en las posibilidades, más deslumbrante aparece. Y para cualquier lector valiente o lo suficientemente aburrido como para leer la documentación hasta el final, el esquema global está allí, presentado en el párrafo final.

Todo empezaba a encajar. Con el aire de inevitabilidad que revela una auténtica obra maestra, en toda la gloria del estilo inimitable del documento, las últimas seis líneas repentinamente dieron sentido a los centenares de páginas anteriores:

> Hay varios cursos públicos, y tenemos conocimiento de que están siendo escritos varios otros libros, que se ocupan de distintos aspectos de UML, todos basados en nuestras publicaciones preliminares previas. Esperamos una amplia difusión de herramientas de soporte, cursos de capacitación y uso de consultores en el futuro. Alentamos fuertemente este tipo de soporte **[¡Bien por ustedes al alentar el soporte de sus propios productos! ¡Qué desinteresado!]** y trabajaremos conjuntamente con autores, capacitadores y consultores para asegurar que sus consultas sean atendidas, de manera de lograr una difusión y soporte consistentes para UML.

A la espera de una respuesta favorable a mi pedido,

Respetuosamente suyo,

Cándido Smith

Referencias

[1] Rational Software Corporation: *0.8 version of the Unified Method: Notation Summary*, en http://www.rational.com.

[2] Rational Software Corporation: *0.91 Addendum to the Unified Modeling Language*, en http://www.rational.com.

[3] Kim Waldén and Jean-Marc Nerson: *Seamless Object-Oriented Software Architecture: Analysis and Design of Reliable Systems*, Prentice Hall, 1995.