Traducido del inglés al español - www.onlinedoctranslator.com

Capítulo catorce. Mantener la integridad del modelo

Una vez trabajé en un proyecto en el que varios equipos trabajaban en paralelo en un nuevo sistema importante. Un día, el equipo que trabajaba en el módulo de facturación al cliente estaba listo para implementar un objeto al que llamaron **Cargar**, cuando descubrieron que otro equipo ya había construido uno. Con diligencia, se propusieron reutilizar el objeto existente. Descubrieron que no tenía un "código de gastos", por lo que agregaron uno. Ya tenía el atributo "monto publicado" que necesitaban. Habían estado planeando llamarlo "monto adeudado", pero¿lo que hay en un nombre?lo cambiaron. Añadiendo algunos métodos y asociaciones más, obtuvieron algo que se parecía a lo que querían, sin alterar lo que había allí. Tuvieron que ignorar muchas asociaciones que no necesitaban, pero su módulo de aplicación se ejecutó.

Unos días más tarde, surgieron misteriosos problemas en el módulo de la aplicación de pago de facturas para los que el **Cargar** había sido escrito originalmente. Extraño **Cargos** Parecía que nadie recordaba haber entrado y eso no tenía ningún sentido. El programa comenzó a fallar cuando se usaron algunas funciones, particularmente el informe de impuestos del mes hasta la fecha. La investigación reveló que el accidente se produjo cuando se utilizó una función que sumaba el monto deducible de todos los pagos del mes actual. Los registros misteriosos no tenían ningún valor en el campo "porcentaje deducible", aunque la validación de la aplicación de ingreso de datos lo requería e incluso ponía un valor predeterminado.

El problema era que estos dos grupos *diferentes modelos*, pero no se dieron cuenta y no había ningún proceso para detectarlo. Cada uno hizo suposiciones sobre la naturaleza de un cargo que fue útil en su contexto (facturar a los clientes versus pagar a los proveedores). Cuando su código se combinó sin resolver estas contradicciones, el resultado fue un software poco confiable.

Si tan solo hubieran sido más conscientes de esta realidad, podrían haber decidido conscientemente cómo lidiar con ella. Eso podría haber significado trabajar juntos para elaborar un modelo común y luego escribir un conjunto de pruebas automatizado para evitar futuras sorpresas. O simplemente podría haber significado un acuerdo para desarrollar modelos separados y mantenerse alejados del código de los demás. De cualquier manera, comienza con un acuerdo explícito sobre los límites dentro de los cuales se aplica cada modelo.

¿Qué hicieron una vez que supieron del problema? Ellos crearon separados**Cargo al cliente** y **Cargo del proveedor** clases y definidas cada una de acuerdo a las necesidades del equipo correspondiente. Resuelto el problema inmediato, volvieron a hacer las cosas como antes. Oh bien.

Aunque rara vez lo pensamos explícitamente, el requisito más fundamental de un modelo es que sea internamente coherente; que sus términos siempre tienen el mismo significado y que no contienen reglas contradictorias. La consistencia interna de un modelo, tal que cada término es inequívoco y ninguna regla se contradice, se denomina*unificación*. Un modelo no tiene sentido a menos que sea lógicamente coherente. En un mundo ideal, tendríamos un modelo único que abarcaría todo el dominio de la empresa. Este modelo estaría unificado, sin definiciones de términos contradictorias o superpuestas. Cada declaración lógica sobre el dominio sería coherente.

Pero el mundo del desarrollo de grandes sistemas no es el mundo ideal. Mantener ese nivel de unificación en todo un sistema empresarial es más problemático de lo que vale la pena. Es necesario permitir que se desarrollen múltiples modelos en diferentes partes del sistema, pero debemos tomar decisiones cuidadosas sobre qué partes del sistema podrán divergir y cuál será su relación entre sí. Necesitamos formas de mantener estrechamente unificadas partes cruciales del modelo. Nada de esto ocurre por sí solo o por buenas intenciones. Sucede solo a través de decisiones de diseño conscientes y la institución de procesos específicos.**La unificación total del modelo de dominio para un sistema grande no será factible ni rentable.**

A veces, la gente lucha contra este hecho. La mayoría de la gente ve el precio que cobran varios modelos al limitar la integración y hacer que la comunicación sea engorrosa. Además de eso, tener más de un modelo de alguna manera parece poco elegante. Esta resistencia a múltiples modelos conduce a veces a intentos muy ambiciosos de unificar todo el software en un gran proyecto bajo un solo modelo. Sé que he sido culpable de este tipo de extralimitación. Pero considere los riesgos.

1. **Se pueden intentar demasiados reemplazos heredados a la vez.**

-

Los proyectos grandes pueden empantanarse porque la coordinación sobrepasa sus capacidades.

-

Las aplicaciones con requisitos especializados pueden tener que usar modelos que no satisfacen completamente sus necesidades, lo que las obliga a colocar el comportamiento en otra parte.

-

Por el contrario, intentar satisfacer a todos con un solo modelo puede conducir a opciones complejas que hacen que el modelo sea difícil de usar.

Es más, es probable que las divergencias en los modelos provengan de la fragmentación política y las diferentes prioridades de gestión que de preocupaciones técnicas. Y la aparición de diferentes modelos puede ser el resultado de la organización del equipo y el proceso de desarrollo. Entonces, incluso cuando ningún factor técnico impida la integración completa, el proyecto aún puede enfrentar múltiples modelos.

Dado que no es factible mantener un modelo unificado para toda una empresa, no tenemos que dejarnos a merced de los eventos. Mediante una combinación de decisiones proactivas sobre lo que debe ser unificado y el reconocimiento pragmático de lo que no está unificado, podemos

Cree una imagen clara y compartida de la situación. Con eso en la mano, podemos asegurarnos de que las partes que queremos unificar permanezcan así, y las partes que no están unificadas no causen confusión o corrupción.

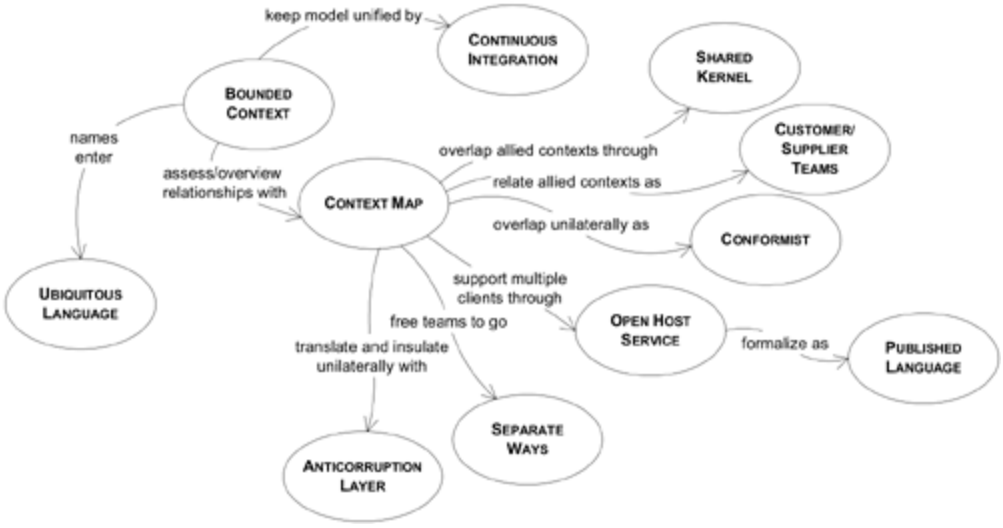
Necesitamos una forma de marcar los límites y las relaciones entre diferentes modelos. Necesitamos elegir nuestra estrategia conscientemente y luego seguir nuestra estrategia de manera consistente.

Este capítulo presenta técnicas para reconocer, comunicar y elegir los límites de un modelo y sus relaciones con otros. Todo comienza con el mapeo del terreno actual del proyecto. ACONTEXTO LIMITADO define el rango de aplicabilidad de cada modelo, mientras que un MAPA DE CONTEXTO ofrece una visión global de los contextos del proyecto y las relaciones entre ellos. Esta reducción de la ambigüedad cambiará, en sí misma, la forma en que suceden las cosas en el proyecto, pero no es necesariamente suficiente. Una vez que tenemos unCONTEXTO LIMITADO, un proceso de INTEGRACIÓN CONTINUA mantendrá el modelo unificado.

Entonces, a partir de esta situación estable, podemos comenzar a migrar hacia estrategias más efectivas para CONTEXTOS DELIMITANTES y relacionarlos, que van desde contextos estrechamente aliados con GRANOS COMPARTIDOS a modelos poco acoplados que van a su CAMINOS SEPARADOS.

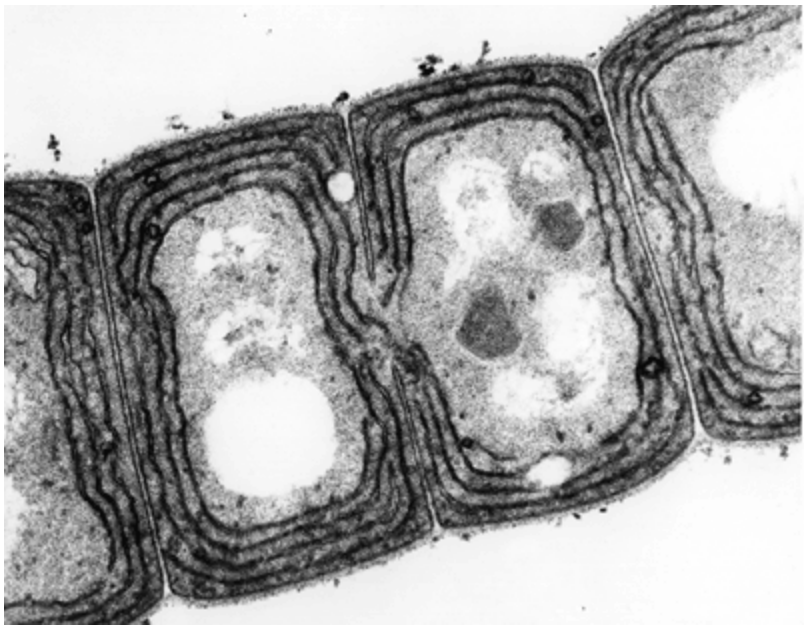
Figura 14.1. Un mapa de navegación para la integridad del modelo

patrones


keep model unified by
CONTINUOUS INTEGRATION
BOUNDED CONTEXT
names enter
UBIQUITOUS LANGUAGE
assess/overview relationships with
CONTEXT MAP
overlap allied contexts through
SHARED KERNEL
relate allied contexts as
CUSTOMER/ SUPPLIER TEAMS
overlap unilaterally as
CONFORMIST
support multiple clients through
OPEN HOST SERVICE
formalize as
PUBLISHED LANGUAGE
free teams to go
translate and insulate unilaterally with
SEPARATE WAYS
ANTICORRUPTION LAYER

Contexto limitado

Las células pueden existir porque sus membranas definen lo que entra y sale y determinan lo que puede pasar.



Varios modelos coexisten en grandes proyectos y esto funciona bien en muchos casos. Se aplican diferentes modelos en diferentes contextos. Por ejemplo, es posible que deba integrar su nuevo software con un sistema externo sobre el que su equipo no tiene control. Una situación como esta probablemente sea clara para todos como un contexto distinto donde el modelo en desarrollo no se aplica, pero otras situaciones pueden ser más vagas y confusas. En la historia que abrió este capítulo, dos equipos estaban trabajando en diferentes funcionalidades para el mismo nuevo sistema. ¿Estaban trabajando en el mismo modelo? Su intención era compartir al menos parte de lo que hacían, pero no había una demarcación para decirles lo que compartían o no. Y no tenían ningún proceso para mantener un modelo compartido unido o detectar rápidamente divergencias.

Incluso un solo equipo puede terminar con varios modelos. La comunicación puede fallar, dando lugar a interpretaciones sutilmente conflictivas del modelo. El código más antiguo a menudo refleja una concepción anterior del modelo que es sutilmente diferente del modelo actual.

Todo el mundo es consciente de que el formato de datos de otro sistema es diferente y requiere una conversión de datos, pero esta es solo la dimensión mecánica del problema. Más fundamental es la diferencia en los modelos implícitos en los dos sistemas. Cuando la discrepancia no es con un sistema externo, sino dentro de la misma base de código, es incluso menos probable que se reconozca. Sin embargo, esto sucede en *todos* grandes proyectos de equipo.

Hay varios modelos en juego en cualquier proyecto grande. Sin embargo, cuando se combina código basado en distintos modelos, el software se vuelve defectuoso, poco confiable y difícil de entender. La comunicación entre los miembros del equipo se vuelve confusa. A menudo no está claro en qué contexto un modelo debe *no* se aplicado.

La falla en mantener las cosas en orden se revela en última instancia cuando el código en ejecución no funciona correctamente, pero el problema comienza en la forma en que los equipos están organizados y la forma en que las personas interactúan. Por lo tanto, para aclarar el contexto de un modelo, tenemos que mirar tanto el proyecto como sus productos finales (código, esquemas de base de datos, etc.).

Un modelo se aplica en un *contexto.* El contexto puede ser una determinada parte del código o el trabajo de un equipo en particular. Para un modelo inventado en una sesión de lluvia de ideas, el contexto podría limitarse a esa conversación en particular. El contexto de un modelo utilizado en un ejemplo de este libro es esa sección de ejemplo en particular y cualquier discusión posterior al respecto. El contexto del modelo es cualquier conjunto de condiciones que se deben aplicar para poder decir que los términos en un modelo tienen un significado específico.

Para comenzar a resolver los problemas de múltiples modelos, necesitamos definir explícitamente el alcance de un modelo en particular como una parte acotada de un sistema de software dentro del cual se aplicará un solo modelo y se mantendrá lo más unificado posible. Esta definición debe conciliarse con la organización del equipo.

Por lo tanto:

Defina explícitamente el contexto dentro del cual se aplica un modelo. Establezca límites explícitamente en términos de organización del equipo, uso dentro de partes específicas de la aplicación y manifestaciones físicas como bases de código y esquemas de bases de datos. Mantenga el modelo estrictamente consistente dentro de estos límites, pero no se distraiga ni se confunda con problemas externos.

A CONTEXTO LIMITADO delimita la aplicabilidad de un modelo en particular para que los miembros del equipo tengan un entendimiento claro y compartido de lo que tiene que ser consistente y cómo se relaciona con otros CONTEXTOS. Dentro de eso CONTEXTO, trabaje para mantener el modelo lógicamente unificado, pero no se preocupe por la aplicabilidad fuera de esos límites. En otraCONTEXTOS, se aplican otros modelos, con diferencias en terminología, en conceptos y reglas, y en dialectos del LENGUAJE UBIQUITO. Al trazar un límite explícito, puede mantener el modelo puro y, por lo tanto, potente, donde sea aplicable. Al mismo tiempo, evita confusiones al centrar su atención en otrosCONTEXTOS. La integración a través de los límites implicará necesariamente alguna traducción, que puede analizar explícitamente.

BOUNDED CONTEXTOS No son MÓDULAS

Los problemas se confunden a veces, pero se trata de patrones diferentes con diferentes motivaciones. Es cierto que cuando se reconoce que dos conjuntos de objetos forman modelos diferentes, casi siempre se colocan enMÓDULOS. Si lo hace, proporciona diferentes espacios de nombres (esenciales para diferentesCONTEXTOS) y alguna demarcación.

Pero MÓDULOS también organiza los elementos dentro de un modelo; no necesariamente comunican la intención de separarseCONTEXTOS. Los espacios de nombres separados que MÓDULOS crear *dentro de* a CONTEXTO LIMITADO en realidad, dificultan la detección de la fragmentación accidental del modelo.

Ejemplo

Contexto de reserva

Una naviera tiene un proyecto interno para desarrollar una nueva aplicación para reservar carga. Esta aplicación debe ser impulsada por un modelo de objetos. Cuál es elCONTEXTO LIMITADO dentro del cual se aplica este modelo? Para responder a esta pregunta, tenemos que mirar lo que está sucediendo en el proyecto. Tenga en cuenta que este es un vistazo al proyecto.*como están las cosas*, no como debería ser idealmente.

Un equipo de proyecto está trabajando en la propia aplicación de reserva. No se espera que modifiquen los objetos del modelo, pero la aplicación que están construyendo tiene que mostrar y manipular esos objetos. Este equipo es consumidor del modelo. El modelo es válido dentro de la aplicación (su consumidor principal) y, por lo tanto, la aplicación de reserva está dentro de los límites.

Las reservas completadas deben pasarse al sistema de seguimiento de vehículos heredado. Se tomó la decisión por adelantado de que el nuevo modelo se apartaría del modelo heredado, por lo que el sistema de seguimiento de vehículos heredado está fuera de los límites. La traducción necesaria entre el nuevo modelo y el legado será responsabilidad del equipo de mantenimiento del legado. El mecanismo de traducción no es

impulsado por el modelo. No está en elCONTEXTO LIMITADO. (Es parte del límite mismo, que se discutirá enMAPA DE CONTEXTO.) Es bueno que la traducción esté fuera de CONTEXTO (no basado en el modelo). No sería realista pedirle al equipo heredado que hiciera un uso real del modelo porque su trabajo principal está fuera deCONTEXTO.

El equipo responsable del modelo se ocupa de todo el ciclo de vida de cada objeto, incluida la persistencia. Debido a que este equipo tiene el control del esquema de la base de datos, ha estado manteniendo deliberadamente el mapeo relacional de objetos sencillo. En otras palabras, el esquema está siendo impulsado por el modelo y, por lo tanto, está dentro de los límites.

Otro equipo está trabajando en un modelo y una aplicación para programar los viajes de los buques de carga. Los equipos de programación y reserva se iniciaron juntos, y ambos equipos tenían la intención de producir un sistema único y unificado. Los dos equipos se han coordinado casualmente entre sí y ocasionalmente comparten objetos, pero no son sistemáticos al respecto. Son *no* trabajando en el mismo
CONTEXTO LIMITADO. Esto es un riesgo, porque no creen que trabajen en modelos separados. En la medida en que se integren, habrá problemas a menos que pongan en marcha procesos para gestionar la situación. (LosNÚCLEO COMPARTIDO, discutido más adelante en este capítulo, podría ser una buena opción.) El primer paso, sin embargo, es reconocer la situación *como están las cosas*. No estan en lo mismoCONTEXTO y debería dejar de intentar compartir código hasta que se realicen algunos cambios.

Esta CONTEXTO LIMITADO se compone de todos aquellos aspectos del sistema que son impulsados por este modelo en particular: los objetos del modelo, el esquema de la base de datos que persiste los objetos del modelo y la aplicación de reserva. Dos equipos trabajan principalmente en esteCONTEXTO: el equipo de modelado y el equipo de aplicación. La información debe intercambiarse con el sistema de seguimiento heredado, y el equipo heredado tiene la responsabilidad principal de la traducción en este límite, con la cooperación del equipo de modelado. No existe una relación claramente definida entre el modelo de reserva y el modelo de programación de viajes, y definir esa relación debería ser una de las primeras acciones de esos equipos.

Mientras tanto, deben tener mucho cuidado al compartir código o datos.

Entonces, ¿qué se ha ganado al definir este CONTEXTO LIMITADO? Para los equipos que trabajan enCONTEXTO: claridad. Esos dos equipos saben que deben ser consistentes con un modelo. Toman decisiones de diseño con ese conocimiento y están atentos a las fracturas. Para los equipos de fuera: libertad. No tienen que caminar en la zona gris, sin usar el mismo modelo, pero de alguna manera sienten que deberían hacerlo. Pero la ganancia más concreta en este caso particular es probablemente darse cuenta del riesgo del intercambio informal entre el equipo del modelo de reserva y el equipo del cronograma del viaje. Para evitar problemas, realmente necesitan decidir sobre las compensaciones de costo / beneficio de compartir e implementar procesos para que funcione. Esto no sucederá a menos que todos comprendan dónde están los límites de los contextos del modelo.

❄ ❄ ❄

Por supuesto, los límites son lugares especiales. Las relaciones entre un CONTEXTO LIMITADO y sus vecinos requieren cuidados y atención. los MAPA DE CONTEXTO traza el territorio, dando el panorama general de la CONTEXTOS y sus conexiones, mientras que varios patrones definen la naturaleza de las diversas relaciones entre CONTEXTOS. Y un proceso de INTEGRACIÓN CONTINUA preserva la unidad del modelo dentro de un CONTEXTO LIMITADO.

Pero antes de pasar a todo eso, ¿cómo se ve cuando se rompe la unificación de un modelo? ¿Cómo reconoces las astillas conceptuales?

Reconociendo astillas dentro de una BOUNDED CONTEXT

Muchos síntomas pueden indicar diferencias de modelo no reconocidas. Algunos de los más obvios son cuando las interfaces codificadas no coinciden