عنوان: **(Domain Driven Design) به زبان ساده**
نویسنده: دلپاک
تاریخ: ۱۳۹۲/۰۴/۰۹ ۱۸:۵۰
آدرس: www.dotnettips.info
برچسب‌ها: Domain Driven Design, DDD

DDD چیست؟ روشی است ساده، زیبا، در وهله اول برای تفکر، و در وهله دوم برای توسعه نرم افزار، که می‌توان بر مبنای آن نیازمندیهای پویا و پیچیده‌ی حوزه دامین را تحلیل، مدل و نهایتا پیاده سازی کرد.
در این روش توسعه نرم افزار تاکید ویژه ای بر الزامات زیر وجود دارد:

تمرکز اصلی پروژه، باید صرف فائق آمدن بر مشکلات و پیچیدگیهای موجود در دامین شود.
پیچیدگیهای موجود در دامین پس از شناسایی به یک مدل تبدیل شوند.
برقراری یک رابطه‌ی خلاق بین متخصصان دامین و افراد تیم توسعه برای بهبود مستمر مدل ارائه شده که نهایتا راه حل مشکلات دامین است بسیار مهم می‌باشد.

مبدع این روش کیست؟
بخوانید:
Eric Evans is a specialist in domain modeling and design in large business systems. Since the early 1990s, he has worked on many projects developing large business systems with objects and has been deeply involved in applying Agile processes on real projects. Eric is the author of "Domain-Driven Design" (Addison-Wesley, 2003) .and he leads the consulting group Domain Language, Inc
ویدیویی سخنرانی اریک اوانس با عنوان: **DDD: putting the model to work** http://www.infoq.com/presentations/model-to-work-evans
شرایط موفقیت اجرای یک پروژه مبتنی بر DDD چیست؟

دامین ساده و سر راست نباشد.
افراد تیم توسعه با طراحی / برنامه نویسی شی گرا آشنا باشند.
دسترسی به افراد متخصص در مسائل مرتبط با دامین آسان باشد.
فرآیند تولید نرم افزار، یک فرآیند چابک باشد.

در این باره چه چیزی بخوانم؟
برای شروع توصیه می‌کنم بخوانید: *Domain-Driven Design: Tackling Complexity in the Heart of Software*
نویسنده: Eric Evans
لینک: http://www.amazon.com/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215

در ادامه می‌پردازم به اینکه ابزار DDD برای شکستن پیچیدگیهای دامین و تبدیل آنها به مدل کدامند؟
پاسخ خلاصه در زیر آمده است. دعوت می‌کنم تا شرح هر یک از این موارد را در پستهای بعدی پیگیر باشید.

Entity
Value Object
Aggregate
Service
Repository
Factory

قبلا توضیح داده بودم که طراحی متاثر از حوزه‌ی کاری (Domain Driven Design) منجر به کاهش، درک و نهایتا قابل مدیریت کردن پیچیدگیهای موجود در حوزه عملیاتی نرم افزار (Domain) می‌شود. توجه کنیم که تحلیل و مدلسازی، فعالیتهای رایج در حوزه هایی مانند حمل و نقل، اکتشافات، هوا فضا، شبکه‌های اجتماعی و ... می‌تواند بسیار دشوار باشد.

برای مثال فرض کنید که می‌خواهید به مدلسازی نرم افزاری بپردازید که هدف تولید آن پیش بینی وضع آب و هوا است. این حوزه کاری (پیش بینی وضع آب و هوا) بویژه همواره یکی از حوزه (Domain)های پیچیده برای طراحان مدل محسوب می‌شود. DDD روشی است که با بکار گیری آن می‌توانید این پیچیدگیها را بسیار کاهش دهید. توسعه سیستمهای پیچیده بدون رعایت قواعدی همه فهم، نهایتا نرم افزار را به مجموعه ای از کدهای غیر خوانا و دیرفهم تبدیل می‌کند که احتمالا نسل بعدی توسعه دهندگانش را تشویق به بازنویسی آن خواهد کرد.

DDD در واقع روشی است برای دقیق‌تر فکر کردن در مورد نحوه ارتباط و تعامل اجزای مدل با یکدیگر. این سبک طراحی (DDD) به تولید مدلی از اشیاء می‌انجامد که نهایتا تصویری قابل درک (Model) از مسائل مطرح در حوزه‌ی کاری ارائه می‌دهند. این مدل ارزشمند است وقتی که:

-1 نمایی آشکار و در عین حال در نهایت سادگی و وضوح از همه‌ی مفاهیمی باشد که در حوزه عملیاتی نرم افزار(Domain) وجود دارد.
-2 به تناسب درک کاملتری که از حوزه کاری کسب می‌شود بتوان این مدل را بهبود و توسعه داد(Refactoring).

در DDD کوشش می‌شود که با برقراری ارتباطی منطقی بین اشیاء، و رعایت سطوحی از انتزاع یک مشکل بزرگتر را به مشکلات کوچکتر شکست و سپس به حل این مشکلات کوچکتر پرداخت.

توجه کنیم که DDD به چگونگی سازماندهی اشیاء توجه ویژه ای دارد ولی DDD چیزی درباره برنامه نویسی شی گرا نیست. DDD متدولوژیی است که با بهره گیری از مفاهیم شی گرایی و مجموعه ای از تجارب ممتاز توسعه نرم افزار (Best Practice) پیچیدگیها را و قابلیت توسعه و نگهداری نرم افزار را بهبود می‌دهد.

ایده مستتر در DDD ساده است. اگر در حوزه کاری مفهوم (Concept) ارزشمندی وجود دارد باید بتوان آن را به وضوح در مدل مشاهده کرد. برای مثال صحیح نیست که یک شرط ارزشمند حوزه کسب و کار را با مجموعه‌ی سخت فهمی از If / Else ها، در مدل نشان داد. این شرط مهم را می‌توان با Specification pattern پیاده سازی کرد تا تصویری خواناتر از Domain در مدل بوجود بیاید.

مدلی که دستیابی به آن در DDD دنبال می‌شود مدلی است سر راست و گویا با در نظر گرفتن همه جوانب و قواعد حوزه‌ی عمل نرم افزار. در این مدل مطلوب و ایده آل به مسائل ذخیره سازی (persistence) پرداخته نمی‌شود. (رعایت اصل PI). این مدل نگران چگونگی نمایش داده‌ها در واسط کاربری نیست. این مدل نگران داستانهای Ajax نیست. این مدل یک مدل Pureاست که دوست داریم حتی المقدور POCO باشد. این مدلی است برای بیان قواعد و منطق موجود در حوزه عمل نرم افزار. این قواعد همیشه تغییر می‌کنند و مدلی که از آموزه‌های DDDالهام گرفته باشد، راحت‌تر پذیرای این تغییرات خواهد بود. به همین دلیل DDD با روشهای توسعه به سبک اجایل نیز قرابت بیشتری دارد.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۴/۰۹ ۱۹:۱۲

عنوان مطلب هست «به زبان ساده»، ولی اصل PI و Specification pattern و مانند اون در مقدمه مطلب بکار رفتن که کمی برای قسمت اول زیاده روی هست. ضمنا Refactoring اصلا به معنای بهبود و توسعه همزمان سیستم نیست. Refactoring بهبود کیفیت کدها هست بدون تغییری در ساختار کلی سیستم.

نویسنده: دلپاک
تاریخ: ۱۳۹۲/۰۴/۱۶ ۱۷:۱۰

Re-factoring در کانتکست این مطلب به همان معنی است که عرض کردم ولی اگر موضوع حول تجربیات نگهداشت و توسعه سیستم (خارج از بحث DDD) می‌بود میشد به نحوی که شما اشاره کرده اید، تفسیر کرد. اساسا تفسیر واژگان خارج از فضای کلی بحث، می‌تواند گمراه کننده شود.