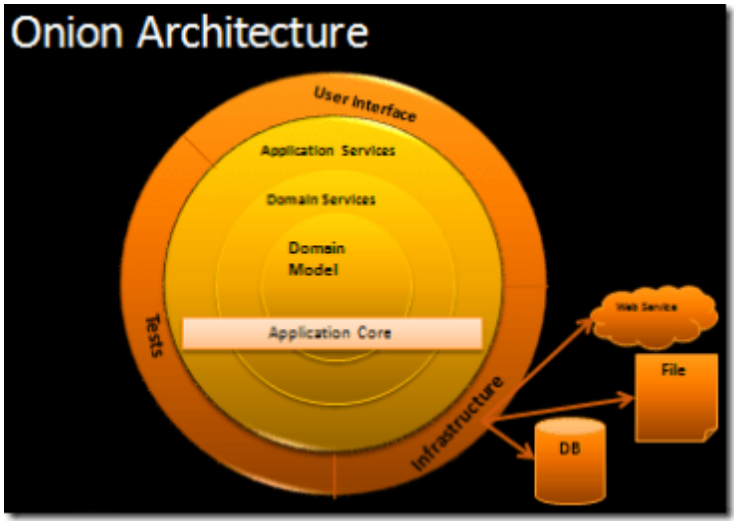
SOFTWARE ENGINEERING

What are application and domain services in onion architecture?

Asked 1 year, 1 month ago Active 1 year, 1 month ago Viewed 2k times

2

Onion architecture has a core which is composed by domain model, domain services and application services:



I'm in doubt about those two service layers, domain services and application services.

I've been reading that they're related to DDD but I'm not familiar with DDD myself.

I'm not asking for the relationship, just an explanation of what those two layers do, and if possible a simple example in Java.

I've read that Domain Services are services used by the domain model and Application Services are services made accessible to the outer layers. Is this correct?

So a Repository would be a Domain Service and Application Services are related to the Use Cases of the application.

All of that is still unclear to me.

java object-oriented onion-architecture

edited Feb 3 '19 at 17:29

asked Feb 3 '19 at 13:19

Piovezan

297 ● 1 ● 11

Something wrong with the question? How can I improve it? – Piovezan Feb 3 '19 at 14:14

2 Answers

Active Oldest Votes

1

> I've read that Domain Services are services used by the domain model ...

In DDD, domain services are part of the domain model/layer. They encapsulate business logic that doesn't neatly fit into a single entity in the model. A classic example is `BankAccount` (entity) and `FundsTransferService` (which operates on two accounts at a time).

> ... and Application Services are services made accessible to the outer layers.

Yes, this is correct. The application service orchestrates the execution of business logic for a use case. A typical application service might:

1. Load related entities from a repository.
2. Execute business logic using operations on entities or via domain services.
3. Store the updated entities.

> So a Repository would be a Domain Service ...

A repository is not a domain service, it simply provides a mechanism to store and query entities but does not house any business logic. The repository interface is still considered part of the domain layer though.

> ... and Application Services are related to the Use Cases of the application.

Yes, however, as mentioned above, they should only orchestrate use cases, not directly implement any business logic.

answered Feb 3 '19 at 18:08

casablanca
2,496 ● 10 ● 14

So the Repository implementation would belong to an outer layer (e.g. infrastructure layer)? And regarding the application services/use case orchestration, I'd like to go into details, do you happen to have a reference I could study? Or a tip on what I should look for? – Piovezan Feb 3 '19 at 18:14

Yes, the repository implementation goes in the infrastructure layer. As for a reference on application services, here's some sample code from Vaughn Vernon's book on DDD: github.com/VaughnVernon/IDDD_Samples – casablanca Feb 3 '19 at 18:42

Regarding DDD's Repository, you're wrong in two ways: 1) often, a Repository will contain complex queries which indeed do contain business logic (if

you ever developed a complex business app, you would know that); 2) The Repository implementation belongs within the Domain layer, *never* the Infrastructure layer (if it was put in Infrastructure, this would create a dependency in the reverse direction, upwards the stack of layers, breaking the fundamental rule of layering, ie, that dependencies only go from higher layers to lower layers). – Rogério Jul 18 '19 at 19:09

@Rogério: I work on very large enterprise applications and no, you don't want to put business logic in your queries. Doing so locks your business logic into a particular database implementation and makes it very hard to switch to a different vendor. This goes beyond DDD, you never want to couple your business logic to an external product. Also I don't understand your argument about layering, the infrastructure layer is the outermost layer and the domain layer does not depend on it. – casablanca Jul 21 '19 at 3:15

Your answer talks about DDD layered architecture, *not* about Onion architecture. And in DDD, we have four layers: UI -> Application -> Domain -> Infrastructure, with source code dependencies going exclusively in the direction of the arrows. See the DDD book if you don't believe me. Regarding business logic in queries, this is inevitable (again, if you ever developed a complex business app you would know), and that "switch to a different vendor" argument is very weak. In practice, we tend to use an ORM (JPA/Hibernate in Java), which helps a lot with that. – Rogério Jul 22 '19 at 14:55

Answer to your question

2

Taking Jeff Palermo's own explanations:

- **Domain Services**: it is interfaces who are implemented in the outer layers, since they need some services of the outer world (i.e. databases).

  > The first layer around the Domain Model is typically where we would find interfaces that provide object saving and retrieving behavior, called ***repository interfaces***. The object saving behavior is not in the application core, however, because it typically involves a database. Only the interface is in the application core.

- **Application Services**: it is the application logic, that doesn't depend on the interfaces to the outside world (so independent of the user interface as well).

  > The code that interacts with the database will implement interfaces in the application core. The application core is coupled to those interfaces but not the actual data access code. In this way, we can change code in any outer layer without affecting the application core.

So this architecture is about dependencies and abstractions, for example:

- in the inner circle, you'll have `IConference`, an abstraction of the domain object.
- in the domain services, you'll have `IConferenceRepository` that will allows you to retrieve and save `IConferences`.
- in the application services, you'll have things like booking of conferences by users. THe application service depends then only on `IConference`, `IConferenceRepository`, `IUser` and `IUserRepository`. This ensures that application code still compiles, whatever the real implementation of the interfaces will be.
- in the outer layers, you'll have the concrete implementation of `ConferenceRepository` that would then use a concrete database to do its job.

Further thoughts on Onion Architecture

First of all, I'd strongly advise that you read Evan's DDD book. It's really worth the investment and the time, because it is well written, it's real-world approach for complex models, and its terminology is well accepted and precise.

Personally, I'll find the Onion Architecture confusing because:

- it uses ***DDD terminology*** like ***Domain Services***, but with a completely different meaning. In DDD, a domain service encapsulates behaviors that do not fit in a single domain object.
- Palermo's pages do not describe very precisely the different parts of his architecture (and the wording is unprecise, since *application services* becomes *application core* in the part where he explains it, but he uses the same term to describe the application service layer and the core composed of the layer and the inner layers).
- The explanations are example based, but the examples are no longer accessible.

If I were you, I'd invest more in Uncle Bob's Clean Architecture that relies on the same good DIP practices, but has a better terminology (the inner is the Entity core, and the applciation core is called "Use Case" which is more illustrative of what's really in). In addition there's a reference book that describes it very well.

answered Feb 3 '19 at 18:42

Christophe

39.5k ● 5 ● 58 ● 97

1 +1 for mentioning Clean Architecture, I agree that Onion Architecture is somewhat confusing. – casablanca Feb 3 '19 at 18:48

Thanks. I wasn't aware Palermo's notes described those layers. – Piovezan Feb 3 '19 at 18:58