Jeffrey Palermo (.com)

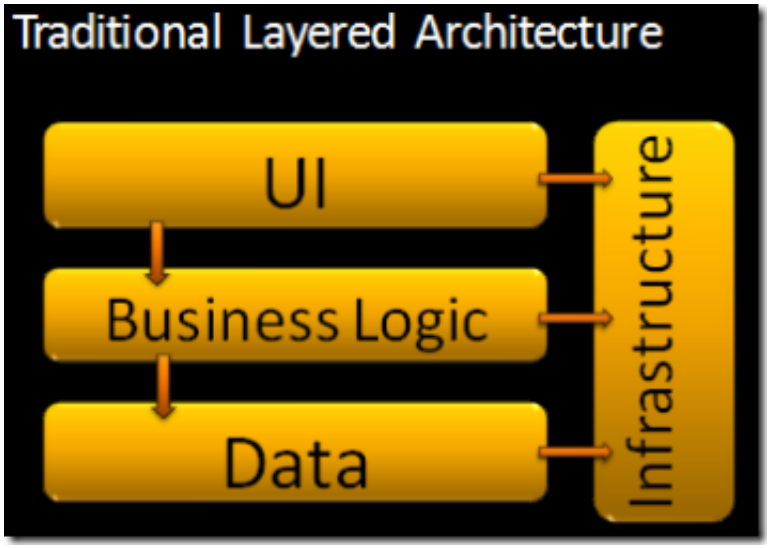
Managing Partner/CEO, Clear Measure

The Onion Architecture : part 1

29 July 2008

This is part 1. part 2. part 3. part 4. My feed (rss).

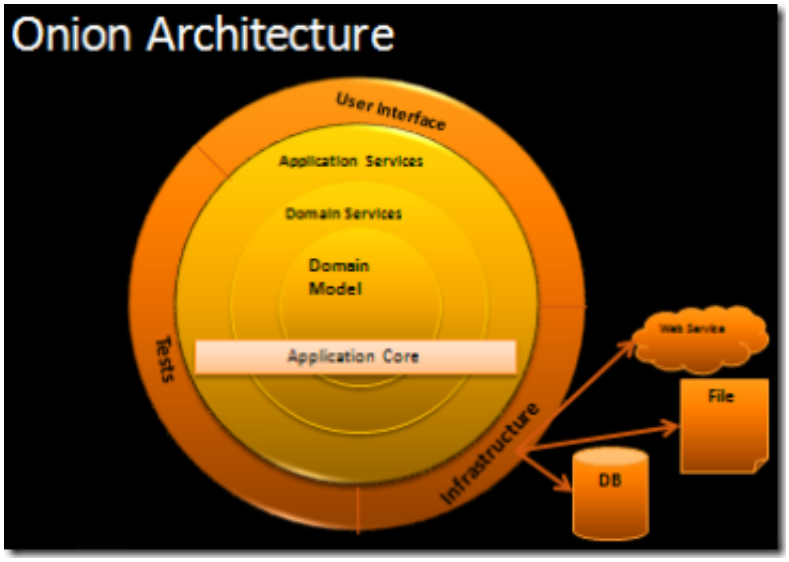
I've spoken several times about a specific type of architecture I call "Onion Architecture". I've found that it leads to more maintainable applications since it emphasizes separation of concerns throughout the system. I must set the context for the use of this architecture before proceeding. This architecture is not appropriate for small websites. It is appropriate for long-lived business applications as well as applications with complex behavior. It emphasizes the use of interfaces for behavior contracts, and it forces the externalization of infrastructure. The diagram you see here is a representation of traditional layered architecture. This is the basic architecture I see most frequently used. Each subsequent layer depends on the layers beneath it, and then every layer normally will depend on some common infrastructure and utility services. The big drawback to this top-down layered architecture is the coupling that it creates. Each layer is coupled to the layers below it, and each layer is often coupled to various infrastructure concerns. However, without coupling, our systems wouldn't do anything useful, but this architecture creates unnecessary coupling.



The biggest offender (and most common) is the coupling of UI and business logic to data access. Yes, UI is coupled to data access with this approach. Transitive dependencies are still dependencies. The UI can't function if business logic isn't there. Business logic can't function if data access isn't there. I'm intentionally ignoring infrastructure here because this typically varies from system to system. Data access changes frequently. Historically, the industry has modified data access techniques at least every three years; therefore, we can count on needing to modify data access three years from now for any healthy, long-lived systems that's mission-critical to the business. We often don't keep systems up-to-date because it's impossible to do. If coupling prevents easily upgrading parts of the system, then the business has no choice but to let the system fall behind into a state of disrepair. This is how legacy

systems become stale, and eventually they are rewritten.

I propose a new approach to architecture. Honestly, it's not completely new, but I'm proposing it as a named, architectural pattern. Patterns are useful because it gives software professionals a common vocabulary with which to communicate. There are a lot of aspects to the Onion Architecture, and if we have a common term to describe this approach, we can communicate more effectively.



The diagram to the left depicts the Onion Architecture. The main premise is that it controls coupling. The fundamental rule is that all code can depend on layers more central, but code cannot depend on layers further out from the core. In other words, all coupling is toward the center. This architecture is unashamedly biased toward object-oriented programming, and it puts objects before all others.

In the very center we see the Domain Model, which represents the state and behavior combination that models truth for the organization. Around the Domain Model are other layers with more behavior. The number of layers in the application core will vary, but remember that the Domain Model is the very center, and since all coupling is toward the center, the Domain Model is only coupled to itself. 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. Out on the edges we see UI, Infrastructure, and Tests. The outer layer is reserved for things that change often. These things should be intentionally isolated from the application core. Out on the edge, we would find a class that implements a repository interface. This class is coupled to a particular method of data access, and that is why it resides outside the application core. This class implements the repository interface and is thereby coupled to it.

The Onion Architecture relies heavily on the Dependency Inversion principle. The application core needs implementation of core interfaces, and if those implementing classes reside at the edges of the application, we need some mechanism for injecting that code at runtime so the application can do something useful.

The database is not the center. It is external. Externalizing the database can be quite a change for some people used to thinking about applications as "database applications". With Onion Architecture, there are no database applications. There are applications that might use a database as a storage service but only though some external infrastructure code that implements an interface which makes sense to the application core. Decoupling the application from the database, file system, etc, lowers the cost of maintenance for the life of the application.

Alistair Cockburn has written a bit about Hexagonal architecture. Hexagonal architecture and Onion Architecture share the following premise: Externalize infrastructure and write adapter code so that the infrastructure does not become tightly coupled.

I'll be writing more about the Onion Architecture as a default approach for building enterprise

applications. I will stay in the enterprise system space and all discussion will reside in that context. This gets even more interesting when there are multiple processes making up a single software system.

Read More 56 Comments

Trackbacks

The Onion Architecture Posted on 7.29.2008 at 8:51 AM

You've been kicked (a good thing) - Trackback from DotNetKicks.com

Onion Architecture Posted on 7.29.2008 at 9:15 AM

Onion Architecture

Elegant Code » The Onion Architecture Posted on 7.29.2008 at 12:42 PM

Pingback from Elegant Code » The Onion Architecture

The Inquisitive Coder - Davy Brion's Blog » Blog Archive » Onion Architecture? Posted on 7.29.2008 at 2:20 PM

Pingback from The Inquisitive Coder - Davy Brion's Blog » Blog Archive » Onion Architecture?

The Onion Architecture Posted on 7.30.2008 at 1:50 AM

The Onion Architecture

Reflective Perspective - Chris Alcock » The Morning Brew #147 Posted on 7.30.2008 at 2:27 AM

Pingback from Reflective Perspective - Chris Alcock » The Morning Brew #147

Dew Drop - July 30, 2008 | Alvin Ashcraft's Morning Dew Posted on 7.30.2008 at 7:06 AM

Pingback from Dew Drop - July 30, 2008 | Alvin Ashcraft's Morning Dew

The Onion Architecture : part 2 Posted on 7.30.2008 at 8:14 AM

In part 1 , I introduced an architectural pattern that I have named "Onion Architecture". The object-oriented design concepts are not new, but I'm pulling together a lot of techniques and conventions into a single pattern and giving it a name. My hope

So you want to learn NHibernate - Part 1/2, Prerequisites (or NHibernate = Marijuana.NET) « HSI Developer Blog Posted on 7.31.2008 at 12:53 PM

Pingback from So you want to learn NHibernate - Part 1/2, Prerequisites (or NHibernate = Marijuana.NET) « HSI Developer Blog

Links of the Week August 2 2008 Posted on 8.01.2008 at 8:10 PM

Another week of great resources for the .NET developer to take in. Amazing how many are out there. I

Weekly Links #12 | GrantPalin.com Posted on 8.03.2008 at 9:07 PM

Pingback from Weekly Links #12 | GrantPalin.com

The Onion Architecture : part 3 Posted on 8.04.2008 at 9:34 AM

Part 1 - Part 2 - This is part 3 - My RSS feed In my previous installments, I described what has become my approach to defining the architecture for an application. Based on feedback, I've modified my diagrams a bit to reduce ambiguity and emphasize key

Weekly Web Nuggets #23 Posted on 8.04.2008 at 9:36 AM

General The Onion Architecture Part 1 : Jeffrey Palermo proposes a new architectural pattern that is based on how components are coupled to each other. The Onion Architecture Part 2 : Jeffrey Palermo continues his presentation of the Onion Architecture

La arquitectura de la cebolla Posted on 8.07.2008 at 9:02 PM

Pingback from La arquitectura de la cebolla

So you want to learn NHibernate - Part 0.5, Prerequisites (or NHibernate = Marijuana.NET) | The Freak Parade Posted on 8.08.2008 at 5:39 PM

Pingback from So you want to learn NHibernate - Part 0.5, Prerequisites (or NHibernate = Marijuana.NET) | The Freak Parade

Training module - Layering Posted on 8.15.2008 at 1:48 PM

Training module - Layering

Architecture with layers, active records, and onions Posted on 8.16.2008 at 2:39 PM

In the 1990s I coded on a few systems where the architecture was that we attached database functionality

The Onion Architecture | Dev @ Work Posted on 8.26.2008 at 12:43 PM

Pingback from The Onion Architecture | Dev @ Work

Translation Tester, part 1 - The problem « ICoder Posted on 9.10.2008 at 5:01 PM

Pingback from Translation Tester, part 1 - The problem « ICoder

Architektur Roundtripp - Ein Wochen-Review Posted on 9.12.2008 at 5:30 AM

Nachdem ich schon vor einigen Monaten immer mal wieder etwas über DDD gehört und gelesen hatte (meine

Why Test Driven Development is a Hard Sell Posted on 10.06.2008 at 4:41 PM

A Good Presentation Two weekends I went to a San Diego Dot Net User Group meeting that featured two presentations

Why Test Driven Development is a Hard Sell Posted on 10.06.2008 at 6:13 PM

A Good Presentation Two weekends I went to a San Diego Dot Net User Group meeting that featured two presentations

Code Duplication tool on CodePlex Posted on 10.14.2008 at 7:15 AM

Just chatting yesterday about quick wins in code reviews and the subject of de-duping code came up. Even

Logical not Physical, Few not Many, Just Do It Posted on 10.20.2008 at 6:21 PM

In both my “Better Domain Driven Design” and “Want SOA? Throw out your Web Services!”

Code Duplication tool on CodePlex Posted on 10.22.2008 at 4:29 AM

Just chatting yesterday about quick wins in code reviews and the subject of de-duping code came up. Even

Sharon’s Blog » Layering Posted on 11.03.2008 at 11:32 PM

Pingback from Sharon’s Blog » Layering

The Myth of Self-Organizing Teams Posted on 11.11.2008 at 8:29 AM

A hot topic in the agile world is "self-organization". The reaction against tight command and control management structures has swayed the pendulum all the way over to chaos. First, I understand that every team is different, and my views are tainted by

Domain Model Repository != DDD Posted on 12.05.2008 at 1:25 AM

Domain Model Repository != DDD

My current architecture Posted on 12.10.2008 at 12:15 PM

For the six or seven of you in the world that actually read my blog, you've noticed that i've been pretty...

My current architecture Posted on 12.10.2008 at 12:52 PM

For the six or seven of you in the world that actually read my blog, you've noticed that i've

Websites tagged "onion" on Postsaver Posted on 12.31.2008 at 3:02 AM

Pingback from Websites tagged "onion" on Postsaver

Extreme Enthusiasm » Blog Archive » The birthday greetings kata Posted on 1.13.2009 at 9:14 AM

Pingback from Extreme Enthusiasm » Blog Archive » The birthday greetings kata

Comments

Peter Ritchie said on 7.29.2008 at 9:33 AM

I'd argue that what you've described as a tiered architecture isn't a tiered architecture, if any and all layers are accessing data directly. What you're describing, although possibly common, is not a tiered architecture but an application devoid of any real architecture. Which is common; but don't call it a tiered architecture.

A traditional tiered architecture isn't about what's coupled to what; its about what uses what. The type of coupling between tiers is completely orthogonal to the architecture. One implemetnation could use direct coupling, another could as easily use indirect coupling: you could use interface-driven design and make access to *a* business logic layer accessible in one direction from the UI layer, and access to *a* data layer accessible in one directly from the BL layer. The implementation of each of those layers could easily be injected at run-time.

A true tiered architecture should be roughly the same as your onion architecture; except that the data layer is in the centre. The business logic layer wraps the data layer, and the UI layer wraps the business logic layer and there is no infrastructure component. With a layered architecture there should always only be one direction of use/coupling between layers (always flowing down, not up) i.e. coupling/use is always towards the centre in tiered as well, if you diagram tiers as circles with lower layers closer to the centre.

The difference I think is important is that the domain layers have no dependency whatsoever on a data layer/interface.

Billy McCafferty said on 7.29.2008 at 10:21 AM

Peter,

There are many many ways to architect an application; all of them with differing pros and cons for rapidity of development, maintainability and complexity. Although Jeffrey has presented his layers as an onion, they're still separated - under the covers - in an orderly and logically tiered manner. To see a working example of this, download the sample code found at http://www.codeplex.com/SharpArchitecture. This architecture uses the dependency inversion (aka - separated interface), described by Jeffrey, to make the domain be the center of the application while externalizing the data access model. This approach is very maintainable, greatly facilitates test-driven development, and is touted by many demi-gods of development including Robert Martin in Agile Software Development and Martin Fowler in Paterns of Enterprise Application Architecture. But when it comes down to it, we each need to decide which architecture is appropriate for the task at hand when balancing maintainability and complexity.

Brett Baggott said on 7.29.2008 at 10:27 AM

For me, the main takeaway from this was: "This architecture is unashamedly biased toward object-oriented programming, and it puts objects before all others." That made it "click" for me. Even though I understand where Peter is coming from, "objects before all others" also helped me to see that Traditional Tier is not the same as Onion Architecture. Looking forward to more on this Jeffrey. Good stuff.

Mendelt Siebenga said on 7.29.2008 at 10:35 AM

I saw a similar concept on a talk about domain driven design some time ago. I think this came from a paper by Rob Vens. The architecture was displayed as a sunflower with the domain model as the center and all the infrastructure as services around it connected to the domain model by adapters or service interfaces. The UI was seen as infrastructure too.

If I look back at most of the software i've written most of the time this is the architecture that has somehow appeared in the end even though we tried to think in tiers and layers.

I'm not really sure if the application services is a layer in the onion or if it's a separate entity orchestrating the whole. Most of the time the inner layers don't know about the outer layers except through interfaces. And most of the time in my applications the application services are the thing that bind together the domain and the outer layers. But that may be bad design on my part :-)

jpalermo said on 7.29.2008 at 11:18 AM

@Peter,

I am not speaking about tiers. I'm specifically discussing layering. You still have plenty of options for which code runs in each process (tiers), but this architecture only addresses layers, not tiers.

I specifically and intentionally don't put data at the center of the architecture. The database is an infrastructure service that the application makes use of. The object model is the center of the application's universe.

I'm also not describing or advocating commonly-accepted layered architecture. I'm departing from the mainstream and proposing a new architectural approach. Not new in concept, but new as in pattern name.

Ken Egozi said on 7.29.2008 at 12:50 PM

True stuff, well presented.

I'll just add that at the heart of the application, there isn't necessarily *a* Domain Model, but possible *some* Domain Models.

The same application can have several aspects, or contexts, that might be modelled differently.

the cool stuff - all aspects might share the same storage mechanism (i.e. RDBMS tables), and it would fit perfectly in the Domain-driven Onion modelling of the system

John Morales said on 7.29.2008 at 1:54 PM

Maybe in your part 2 you could explain why the tests should be in the external layer. Any change to tests testing core should be a reflection of changing core. Thus if the tests change, it's either a bug, or a business requirement change. For the former I agree on external, but for the latter I need some convincing.

shawn said on 7.29.2008 at 2:38 PM

Jeffrey,

We've been discussing this internally and some questions came up about what you mean when you put tests in the outer layer.

My interpretation is that those are integration tests and are placed there to emphasize that they should only take dependencies against "public APIs" for testing, and that it is implied that each individual component has a suite of tests that take dependencies against nothing else but that component.

Is this correct?

Thanks,

Shawn

Peter Ritchie said on 7.29.2008 at 3:37 PM

Sorry, I threw tier in there when I meant layer.

Bill, I'm not disputing that the onion architecture that Jeff is describing is more agile and scalable than the "other" architecture. I'm just saying the other architecture he's describing is not the recommended "layered" architecture; one where each layer knows nothing of the layer above, each layer implements a specific concern, and no other layers implement concerns that the other layers are implementing.

Yes, the onion architecture is better than the "other" architecture; but that architecture is dysfunctional architecture. Even a correctly implemented traditional three-layer architecture would be better...

jpalermo said on 7.29.2008 at 3:42 PM

@Peter,

I'm completely open for feedback on the "other" layered architecture. My intent is not to put up a straw man only to knock him down. Please advise me how I should revise the traditional diagrams so that it is consistent with how you see traditional layered architecture.

Peter Ritchie said on 7.29.2008 at 4:29 PM

@Jeff: there's three-layer and four-layer architectures. By introducing "Infrastructure" in a layered architecture it makes it sound like four-layer; but you describe it more like three-layer. Four layer has a true infrastructure layer that sits below a domain model layer, which sits below a an application model layer, which sites below a view/UI layer. A three-tier architecture has the UI, BL and DA layers with no infrastructure layer/component. If correctly implemented each layer only knows and talks to the layer beneath it (i.e. with four layer, the only layer that talks to infrastructure is the domain layer). Traditionally these are implemented with direct coupling between layers; but the architectures define usage not coupling--the coupling is an implementation detail. A 3 layer architecture could easily be implemented with the UI only knowing about IBusinessLogic and an IBusinessLogic implementation knowing only of IDataAccess--with different implementations of those interfaces being injected where/when appropriate. While this type of implementation gives you all the the modern testability features we hope for (mocking, separation, etc.), this still essentially leaves a coupling of the business logic layer (although indirect) to a data layer. (IMO, bad)

One of the benefits I think you're you're trying to promote with onion is that the domain layers have no coupling (neither direct, nor indirect) to a data-related implementation and that that data-related implementation isn't one of the layers. But, the way it's described and the way it's diagrammed makes it look like there's just another layer: infrastructure, that lives at the same level as UI.

Tobin Harris said on 7.29.2008 at 5:11 PM

I wish I could communicate that clearly, nice post :)

Necromantici said on 7.29.2008 at 9:44 PM

It would be nice if you made a simple sample of this just to simplify the reasoning of your thoughts.

Shailen Sukul said on 7.30.2008 at 1:22 AM

Nice post that has certainly sparked some good debates. I think that parts of the onion architecture may have been discussed in Martin Fowler's Enterprise Integartion Patterns book (though not specifically called that) and certainly, maintenability is a huge plus, among other things. A .Net demo would be a good next step ... well done

Dave Laribee said on 7.30.2008 at 9:32 AM

I don't see domain services as a layer. They are a concept expressed inside the model layer, along with entities. They will often be verbs in the ubiquitous language or deal with awkward associations between entities or batch/set operations on entities or the need to take dependency on, say, a repository or another domain service. They are infrastructure free, but first class citizens in the model.

Gonzalo Sanchez said on 7.30.2008 at 9:55 AM

I feel this is a consequence of using DSL on your application. I don't see anything new about the "Onion Architecture" but rather a nice layout architecture because DSL.

Evan said on 7.30.2008 at 10:02 AM

Just FYI..

The "Onion" isn't necessarily a new pattern name. It's been around the block a time or two.

stal.blogspot.com/.../onion-model.htm

If you include OS architecture, they also describe their's as an onion:

ou800doc.caldera.com/.../_The_design_of_

That being said, I think the onion metaphor is a great one for describing architecture, and this was a great post. :-)

OtherPeter said on 7.30.2008 at 11:24 AM

Dave, can entities depend on domain services?

Greg Young said on 7.30.2008 at 12:53 PM

I will second Laribee here on the misunderstanding of domain services ... although I think what is happening here is that Jeffrey has just taken a pattern that is widely understood and called something completely different by the same name.

Jeffrey Palermo said on 7.31.2008 at 7:17 AM

@Dave Laribee,

You have a good argument. I should revise the diagram. There is a class of stateless objects that work on entities and domain services, but don't quite fit in the application layer.

Either way, domain-driven design is a big topic in its own right and not meant to be covered in detail in this post.

Fabien Bézagu said on 8.13.2008 at 10:06 AM

Perhaps I'm wrong and I red too fast but it already exists. It's called DDD (Domain Driven Design) and Eric Evans is the father of all patterns related to this approach : www.domaindrivendesign.org

Mihai said on 12.04.2008 at 8:26 AM

The novelty in this pattern seems overrated. I think the novel information here is that the data access "layer" really is not at the center. The decoupling of layers has long been advocated -- DAOs, business delegates, value objects are some products of this intent. The mediators were always supposed to be there to materialize business objects using the DAOs. Where we position the components, how we see these

components as being layered, is what I think was argued here.

I too disagree that the first diagram captures the traditional layered architecture, if the traditional layered architecture is in fact our current ideal of an architecture.

Mihai said on 12.15.2008 at 9:47 AM

On a different note, I think the pattern is the product of the author's independent realization that data sources could be treated as services, their operations mapping to standard DAO operations -- findAllByXXX, save -- (we've had a similar realization during brainstorming at my old workplace and I'm sure many have felt it) and thus deserves credit for speaking up.

Reading around on the layered architecture, there's a key highlight that appears to confirm Leon's diagram as right on the spot: "a layer uses the layer immediately below" is the highlight that sets apart the onion architecture (call it DDD architecture, etc.) from the layered architecture. I wonder where Fowler's dependency injection stands and how it relates the two approaches to each other. I guess DI can go to support either approach.

Jeffrey Palermo said on 12.15.2008 at 10:19 AM

@Mihai,

Thanks for the comment. I have some more information on the Onion Architecture for a part 4. I have to find some time to write it up. I don't want to remain purely in conceptual land, and I want to talk about more than principles, so part 4 will have more concrete implementation information in it. The principles are broad, but implementation rules would also be valuable if communicated.

- **Connect**

- # Recent Blog Entries

  - How to have an open door policy with remote workers?
  - Update on the Clear Measure journey
  - Windows 10 IoT on Raspberry Pi with Visual Studio and C# Universal Apps
  - Code the Town! Investing in the next generation of programmers in Austin, TX
  - Visual Studio 2013 Crashes On Startup–my solution
  - How Windows 10 Ushers in Changes
  - C# is the next-generation cross-platform language
  - Prediction follow-up–Microsoft now licensing Windows Azure to private data centers
  - New release of AliaSQL released to support Everytime scripts
  - Developer interested in business? Join me next week for JWMI event

- # Search

  Search this site

- # Popular Posts

  - Plugged in, not charging Windows 7 solution
  - The Onion Architecture : part 1
  - Adding a second SSD to my Lenovo Yoga & 8GB RAM
  - ASP.NET MVC and the templated partial view (death to ASCX)
  - The Onion Architecture : part 2
  - I am speaking at the Austin Cloud User Group on May 24, 2011 (Tuesday)
  - The Onion Architecture : part 3
  - You should NOT use ASP.NET MVC if. . .
  - I'll get to your application in a minute - First, we need to build the framework
  - How to password-protect in Word 2007

- # Archives

  - April 2016 (2)
  - November 2015 (1)
  - August 2015 (1)
  - June 2015 (2)
  - November 2014 (1)
  - October 2014 (1)

- September 2014 (1)
- August 2014 (1)
- June 2014 (1)
- May 2014 (1)
- February 2014 (1)
- January 2014 (3)
- Older stuff...

Back to Top