Extending Legacy Systems with Domain-Driven Design

A Quick Note

Nigel P. Weymont, Ph.D.

and

Bruce R. Lombardi, Ph.D.

January 2013

Domain-driven Design offers several approaches to building new applications that leverage legacy systems

One of the many strengths of Domain-Driven Design is the opportunity and ability to create well designed new solutions that leverage existing legacy systems. The basic ideas behind this opportunity have been presented by Eric Evans, the father of Domain-Driven Design, at the recent InfoQueue Conference held in New York[1].

The need for new applications is usually driven by business forces, such as new business products or market offerings, changing customer needs, or the need to improve business performance by reducing costs, improving quality, or reducing cycle times. The ability to implement change initiatives is often limited by the constraints imposed by the existing IT infrastructure and the cost and advisability of modifying existing systems.

Modifying legacy systems to provide new features is often difficult and risky

Certainly existing systems could be modified to support new requirements, but this approach often suffers a number of drawbacks. The ability to modify a legacy system may be hampered by a lack of understanding of the inner workings of the system, old and maybe cumbersome technology, and brittle designs and implementations that makes changes risky due to unintended consequences. In Domain-Driven Design parlance, existing systems often conform to the Big Ball of Mud[2] pattern. Modifying a COTS (Commercial Off-the-Shelf) solution can also have many drawbacks, since custom modifications are often difficult to support and update as new versions are rolled out by the supplier, and the modifications have to be re-applied to the new system. The situation is even more complicated if several legacy systems need to be changed in order to deliver the new features that are required. In reality, it is usually so difficult to

to do good design of any kind within legacy systems, that the option to modify legacy systems is often unattractive or even infeasible.

Fortunately, Strategic Domain-Driven Design offers several approaches for working with legacy systems that provide the opportunity to execute a good design, while working within legacy constraints and limitations.

Building a new application allows new thinking, innovation and good design

Building a new application is an attractive alternative for a number of reasons: it allows the domain to be approached differently and incorporate new thinking and perspectives rather than use those inherently embedded in the legacy systems; it allows the business to be more innovative since they are no longer constrained by the legacy systems; and it provides the opportunity to build something that is well designed and outside the constraints of the legacy systems. Nevertheless there are challenges. The new application must have its own bounded context with an accompanying model and Ubiquitous Language. More than likely it is necessary to extract important data from the legacy systems as well as provide new additional data for the new bounded context.

Eric Evans has identified four approaches or strategies for developing new features within a legacy architecture. There are two underlying components to each of the solutions: a new bounded context for the new application, referred to as a "bubble", that is outside the legacy systems; and using the Anti-Corruption Layer[3] (ACL) pattern to translate legacy data into the Ubiquitous Language of the new bounded context. Each of the solutions requires different levels of effort and commitment to

Domain-Driven Design, but deliver correspondingly different capabilities and degrees of sophistication. Importantly, they all provide the possibility of a new, well-designed solution while using the legacy architecture.

The Bubble Context is a simple pattern, uses only legacy data and requires a minimum commitment to Domain-Driven Design

The Bubble Context is the simplest pattern. A new bounded context is defined outside the legacy systems and contains the new domain model. The model is populated with data from the legacy systems by using an ACL that accesses the legacy system and which serves as the repository for the new domain. The level of effort and commitment to Domain-Driven Design is low, but the functionality of the new bounded context is also necessarily limited because all data comes from the legacy systems.

The Autonomous Bubble pattern uses both legacy data and adds its own domain specific data

The Autonomous Bubble pattern elaborates on the Bubble Context and allows the new bounded context to persist its own data as well as interfacing with the legacy systems using an ACL. Clearly more effort is required to implement this pattern, but a more capable domain model is possible. The ACL is, however, single purpose for the new bounded context.

Two patterns are options for opening the legacy systems in a service-oriented way

The next level of sophistication includes the Open Host Services and Synchronizing ACL patterns that expose legacy systems in a service-oriented way and consequently allows the legacy systems to be accessed by multiple, presumably new, bounded contexts. A synchronizing ACL is built on top of the legacy system and provides the services and data structured in a form such as JSON, or possibly some industry specific language. Similar to the Bubble Context and Autonomous Bubble, the data provided by the services can be consumed directly on-demand,

or persisted in its own data store for later consumption. The level of sophistication incorporated in these patterns, such as synchronization between the legacy systems and the new bounded context, requires considerably more effort, but the services are available to multiple clients.

The Event Channel Pattern provides a publish-subscribe approach for accessing legacy systems

The final and most challenging approach is to use an Event Channel pattern. This uses a publish-subscribe design for events that occur within bounded contexts. Events in a bounded context are detected and published for other bounded contexts that are subscribers. The approach is complex and is often not recommended if the other approaches are viable.

Expanding a new bubble requires careful planning and coordination to ensure that emphasis is placed on deriving value from the new domain model

After designing and implementing a new application, careful consideration must be given to planning for the expansion of the bubble. Adding additional information into the domain model because it is available should be avoided unless there is a thoughtful understanding of its relevance to the new domain. Plans that require substantial expansion of the ACL, and provide little additional new value derived from the domain model, should be also be avoided, otherwise disproportionate effort is spent on the ACL, rather than on the domain model which is where business value is created. Iteration plans should focus on a healthy balance between value-adding domain focused work, and the ACL-related work needed to access the legacy systems.

To summarize, new Domain-Driven Design applications can be developed to co-exist with legacy applications by developing the new application in a “bubble” that insulates the new bounded context from the legacy systems. The new bubble interfaces with the legacy system using various forms of an Anti-

Corruption Layer. The approach provides the opportunity to develop new and well-designed applications and a way to begin using Domain-Driven Design. To redesign or refactor legacy systems is unrealistic and unnecessary, but to leverage their value into new applications that are needed for changes in the business, is pragmatic and compelling.

Learn more about Domain-Driven Design

To learn more about Domain-Driven Design and how we may help you, please visit our website at www.domaindrivendesigners.com

References

(1) Eric Evans. "Recovering the Ability to Design When Surrounded by Messy Legacy Systems",QCon New York 2012, http://www.infoq.com/presentations/Strategy-Messy-Legacy-Systems

(2) Brian Foote and Joseph Yoder, *Big Ball of Mud.* Fourth Conference on Patterns Languages of Programs (PLoP '97/EuroPLoP '97) Monticello, Illinois, September 1997

(3) Evans E. Domain-Driven Design, Addison-Wesley, 2004.

About the authors

About Us

We provide comprehensive consulting services for projects either considering, planning, or currently implementing Domain-Driven Design solutions or for project sponsors who are simply looking for better ways of designing and implementing a software solution.

Nigel Weymont has over 25 years experience designing and developing systems for Healthcare, Life Sciences, Chemical Manufacturing, Oil and Gas Processing, Retail, and Financial markets. His experience with Domain-Driven Design arose during development of a large integrated system for use in the Healthcare industry. He is now an enthusiastic proponent of Domain-Driven Design.

+1.888.668.5208

nigel.weymont@domaindrivendesigners.com

Bruce Lombardi has been designing and developing software systems for over 25 years and his experience with Domain-Driven Design began in 2006 while developing a large and complex healthcare application. Bruce and his colleagues discovered that the ideas behind Domain-Driven Design were ideally suited to conquering such complexity. He has been a proponent ever since.

+1.888.668.5208

bruce.lombardi@domaindrivendesigners.com

