(/advanced-topics/)

> *Essentially, all models are wrong, but some are useful.*
>
> —George E. P. Box

Domain Modeling

Domain Modeling is a way to describe and model real world entities and the relationships between them, which collectively describe the problem domain space. Derived from an understanding of system-level requirements, identifying domain entities and their relationships provides an effective basis for understanding and helps practitioners design systems for maintainability, testability, and incremental development. Because there is often a gap between understanding the problem domain and the interpretation of requirements, domain modeling is a primary modeling area in Agile development at scale. Driven in part from object-oriented design approaches, domain modeling envisions the solution as a *set of domain objects that collaborate* to fulfill system-level scenarios.

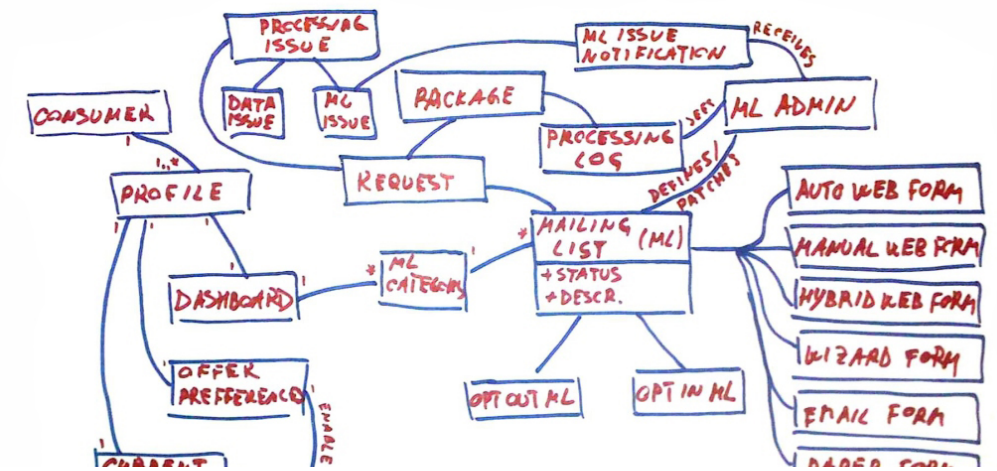
In SAFe, domain modeling connects to backlog items at the Team, Program, Large Solution and Portfolio levels and provides a common language for the entire organization. Especially important is its connection with the Nonfunctional Requirements (NFRs) that may affect certain areas where “alternative design approaches” are sometimes used to satisfy the corresponding NFRs.

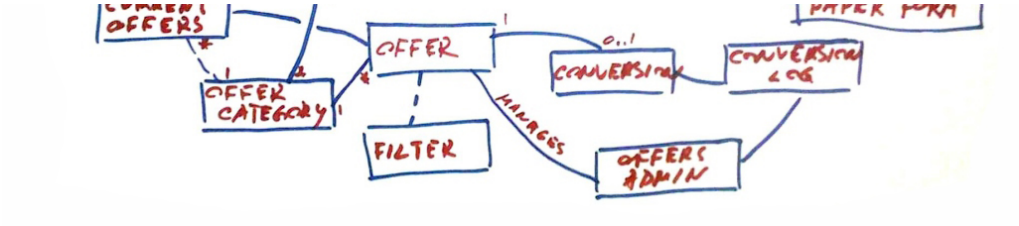
Domain modeling also provides the Agile organization with opportunities for use of Agile-friendly design patterns and approaches that enhance velocity over the long term. As the system design changes, refactoring and updating the domain model is vital to maintaining a continuing understanding of the system, and goes hand in hand with code refactoring to help control the inherent complexity of software systems.

Details

Introduction

Domain modeling is one of the key models used in software engineering: *if you only model one thing in Agile, model the domain*. A relatively small domain-modeling effort is a great tool for controlling the complexity of the system under development. It may help in resolving countless ambiguities in both the requirements and the design intent. Domain modeling simply reflects our understanding of real-world *entities* and their *relationships* and *responsibilities* that cover the problem domain. Figure 1 shows an example of a domain model for a consumer subscription management system:





(https://156s6k36aesz1w6oyx474wo9-wpengine.netdna-ssl.com/wp-content/uploads/2012/11/domain-model-example-figure-1.jpg)
Figure 1. Domain Model for Consumer Subscription Management System

A number of different views or representations express the essential aspects of the problem domain: Robustness Diagram, CRC Cards, ORM Diagram and so on (see [1], chapter 8 for more detail). However, the most simple and common is a class diagram or its simplification (as in Figure 1). Such a diagram primarily shows the key entities and their relations.

Effective domain modeling may only occur in the context of the system-level requirements, often captured as use-cases or other means. In this case, *nouns*, captured from the requirements, become valid candidates for domain entities while *verbs* may represent behaviors and relationships. Together they form a *Common Language* (sometimes called Ubiquitous Language, see [2], chapter 2) that allows engineering, business, and user representatives to speak the same language, minimizing miscommunication.

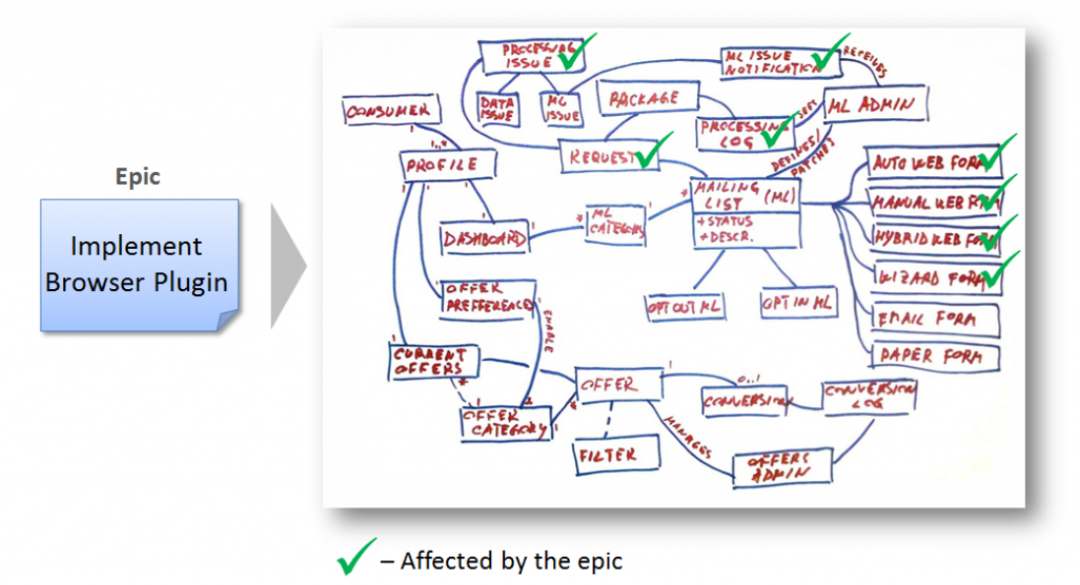
Domain Modeling in Agile at Large Scale

In a large scale Agile development, domain modeling is continuously used to support:

- Analysis of Epics (/epic/)
- Backlog Refinement (/program-and-solution-backlogs/) at Large Solution, Program and Team levels
- Design workshops at different levels
- Refining Vision (/vision/) and Roadmap (/roadmap/) (typically in preparation for Program Increment (/program-increment/))

Domain modeling is typically developed and continuously refined by the System Architect (/system-and-solution-architect-engineering/) in collaboration with other stakeholders in order to understand the impact of epics and features on the system. These groups use domain modeling as part of the preparation for PI Planning at the modeling workshop in a highly visual and collaborative manner.

The following example (Figure 2) shows how a domain model is used to clarify the impact of an epic:



(https://156s6k36aesz1w6oyx474wo9-wpengine.netdna-ssl.com/wp-content/uploads/2012/11/Epics-and-Domain-Model-figure-2.png)
Figure 2. A Domain Model Helps in understanding the scope of an epic

Requirements and domain modeling actually are mutually dependent. Domain modeling supports the clarification of requirements, whereas requirements help to build up and clarifying the model. Furthermore, once new requirements are implemented, the domain model may also change as table 1 suggests.

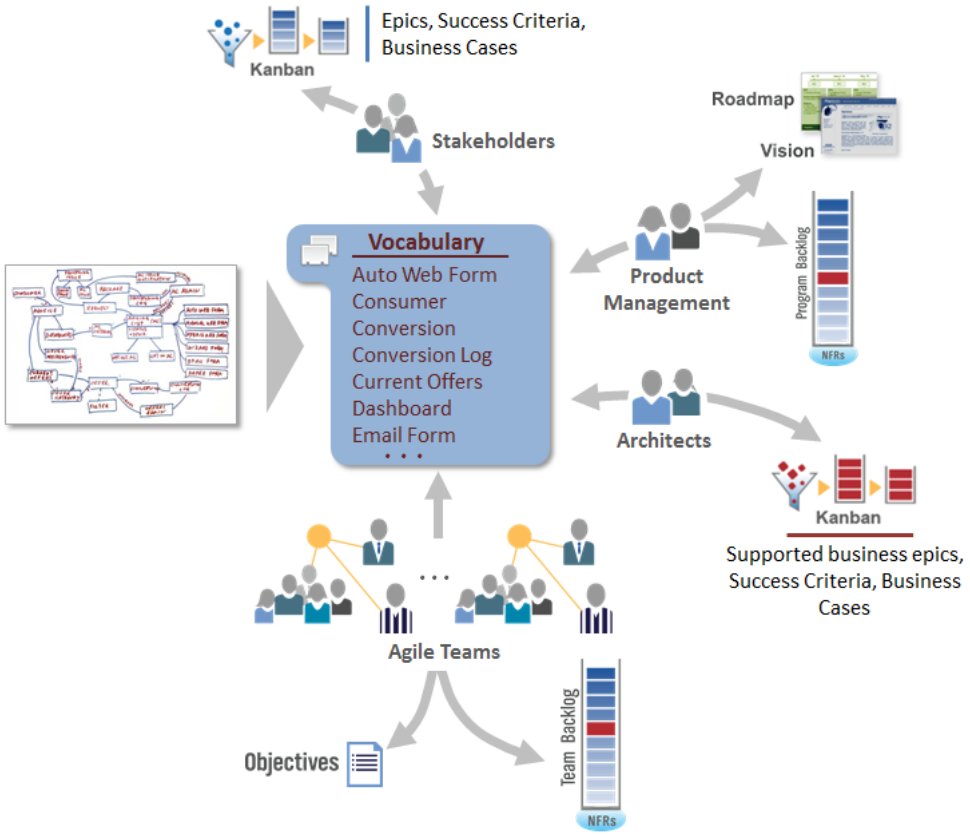
Backlog Item	Impact on Domain Model
Epic (/epic/)	Typically introduces new entities, relationships and responsibilities

Feature (/features-and-capabilities/)	May introduce new entities, typically introduces new relationships or responsibilities
Story (/story/)	Typically introduces changes to the existing relationships and responsibilities, may introduce new responsibilities, value objects or changes to the service interfaces
Enabler Epic (/enablers/)	Typically affects implementation and design aspects of a whole range of entities, services, and repositories such as underlying technology or platform, a generic life cycle of the entities, API constraints etc.E
Enabler Feature (/enablers/)	Introduces changes to implementation/design aspects for a) only certain entities/services typically constrained to one product or system; b) only certain lifecycle steps (e.g.: instantiation). Enabler features may also result in a change of responsibilities or may introduce new value objects.
Refactor (/refactoring/)	May extract individual value objects or "helper" objects out of an entity, may change internal interfaces between entities, protocols or APIs.

Table 1. Impact of SAFe Backlog Items on the Domain Model

Relationships between the entities of the model are critical to effective modeling—without them, the model is just a vocabulary of terms with very broad semantics since they lack their "collaborative" context. Relationships drive both effective requirements definition and design decisions (see the example below in figure 5 and the associated description). Relationships in a domain model can be pretty standard (e.g. 'includes,' 'is a') or very specific (e.g. ML Admin 'defines/patches' the Mailing List in our case). When defining the relationships it is much more important to adequately capture real connections between the entities that convey the meaning of their role rather than to follow format agreements indiscriminately.

The common language resulting from domain modeling is used at all levels of the Agile organization to foster unambiguous shared understanding of the problem domain, requirements, and architecture—see Figure 3.



(https://156s6k36aesz1w6oyx474wo9-wpengine.netdna-ssl.com/wp-content/uploads/2012/11/Common-Language-figure-3.png)
Figure 3. Common Language Used Throughout Agile Organization

Common language—even though is crucial to the product development—has natural limitations that every organization should be aware of. For example, the language of marketing materials may sometimes use terms that diverge from the common language, in order to emphasize certain temporal or subjective aspects associated with current market trends or challenges.

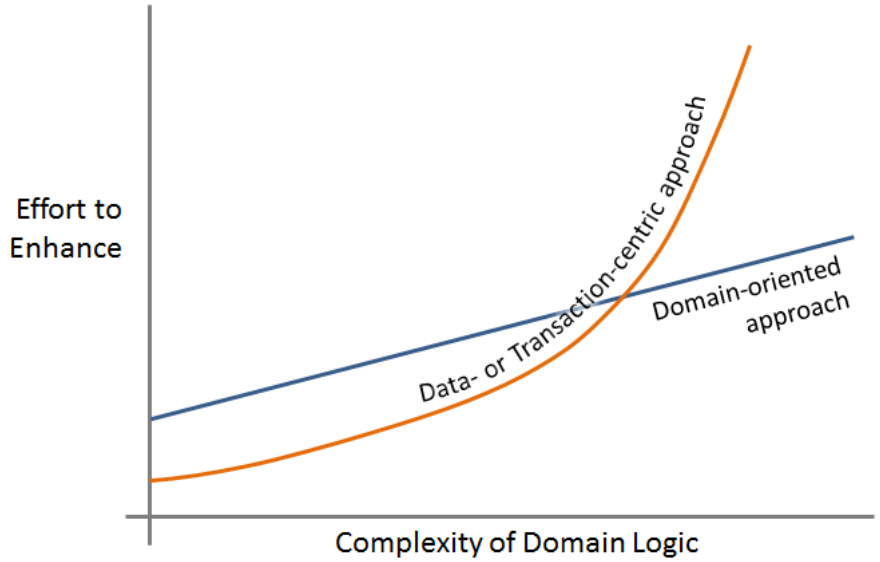
Teams that use Behavior-Driven Development (/behavior-driven-development/) (BDD) inevitably use a common language in their specification workshops when defining human readable tests.

The Domain Model and System Design

Domain-Oriented vs. Alternative Approaches

Domain modeling is not only useful for analysis but is often a good conceptual model for the system design. *Domain modeling* is one of the key design patterns/approaches that assumes *deriving the solution object model directly from the problem domain* while preserving both *behavior* and *data* (see [3]). See [2] for a systematic and detailed outline of such best practices, known by the term of *Domain-Driven Design*. This approach provides a natural and very effective way of managing the inherent complexity of software development that is vital at large scale. Figure 4, adapted from [3], chapter 2, shows a comparison of effort spent on enhancing software functionality versus complexity by different approaches:

- When design is based on the domain
- When design is based on data structure or transaction scripts.

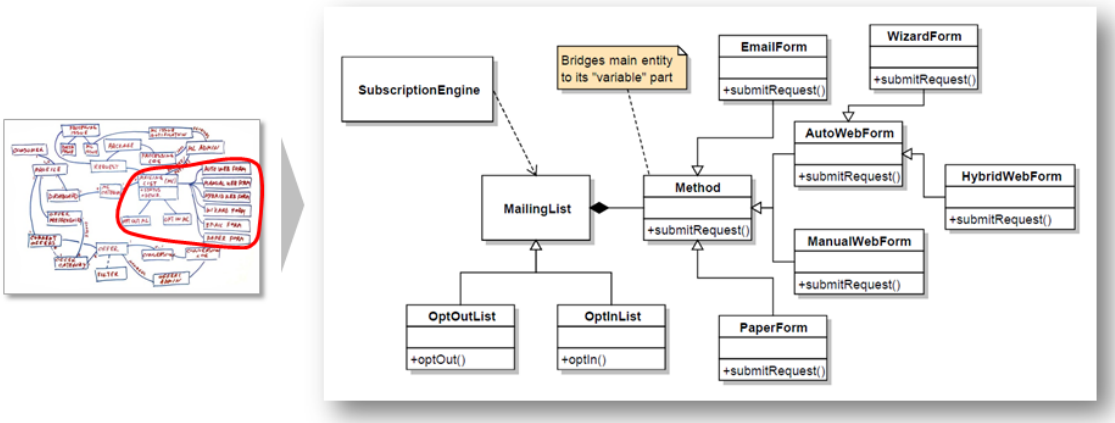


(https://156s6k36aesz1w6oyx474wo9-wpengine.netdna-ssl.com/wp-content/uploads/2012/11/Domain-Oriented-Approach-figure-4.png)

Figure 4. A sense of the relationship between domain-oriented and data- or transaction-centric approaches.

Large-scale software solutions almost inevitably have complex domain logic. Thus data- or transaction-centric design approaches imply a very high cost of maintenance. Nevertheless, too many organizations end up with highly complex system designs that imply a lot of effort to enhance the system. While in some cases such approaches may make sense —and we will discuss those below—most often such a design, in reality, is based on personal preferences of the system architects and teams rather than on business drivers.

One of the many reasons to base system design on the domain structure is to foster reasonable usage of patterns that support maintainability and enable highly incremental, concurrent development. So, in our example of the subscription management system, domain modeling and requirements may logically suggest that subscription methods will represent the primary source of change. Thus, given the different scenarios for opt-in and opt-out functionality, it seems quite logical to use a Bridge Pattern, shown in figure 5 to isolate the area of frequent change and reduce the number of entities in the system—(see [4], Appendix B).



(https://156s6k36aesz1w6oyx474wo9-wpengine.netdna-ssl.com/wp-content/uploads/2012/11/Bridge-Pattern-Derived-From-Domain-Model-figure-5.png)

Figure 5. A bridge pattern derived from analysis of the domain model

This is just one example of how domain modeling can be effectively used for Commonality-Variability Analysis (CVA) to foster effective system object models. (See [5], chapter 8 for more detail on the CVA method).

Domain Modeling, System Design, and Nonfunctional Requirements

Nonfunctional Requirements (/nonfunctional-requirements/), on the other hand, represent the primary reason to build system design around data structure or transaction scripts rather than the domain model. Typically NFRs like performance or scalability may result in cases where domain logic is spread across a bunch of large SQL-scripts, or where too much logic is in the client-side validation scripts, and so on. Even though the use of such a Transaction Script approach (see [3], chapter 9) can be legitimate in certain cases, it should be used as an exception rather than the rule. A very few properly implemented exceptions will still allow the Agile enterprise to benefit from a domain-oriented approach.

Refactoring the Model

Developing a shared understanding with the help of domain modeling is an incremental process just like developing code that implements the underlying domain logic. This means that just like the code, the domain model is also subject to refactoring as our knowledge about the system improves and as new domain entities and their relations actualize, as table 1 suggests. In the Domain-Driven Design approach, keeping the system design and the current understanding of the problem domain up to date is relatively easy, and refactoring of both typically happens synchronously or nearly so (see [2], part III). Designing the effective domain model is both an art and a science. Not uncommonly great insights about the structure and associations in the domain model emerge eventually. However, it is never late to start building the right understanding and to start gradually improving the code towards it—as new functionality allows—to be able to control the complexity.

Summary

Domain modeling is a great tool for Agile enterprise to carry out a common language and a fundamental structure important for the analysis of features and epics. The domain model is defined and continuously refactored as enterprise knowledge about the domain improves and the system functionality evolves. Domain model serves a vital link between the real world where the problem domain resides and the code – domain-oriented design approaches allow to control rapidly growing complexity and cost of maintenance and enhancement effort. Domain modeling is highly collaborative and visual effort that involves system architects, product management, stakeholders and teams all working towards better shared understanding of the priorities and better ways to implement them. There's hardly any other model that would cover that many aspects for agile development at scale. Thus, *if you only model one thing, model the domain.*

Learn More

[1] Ambler, Scott. *Agile Model-Driven Development with UML 2.0.* Cambridge University Press, 2004.

[2] Evans, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison Wesley, 2003.

[3] Fowler, Martin. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002.

[4] Bain, Scott. *Emergent Design: The Evolutionary Nature of Professional Software Development.* Addison Wesley, 2008.

[5] Shalloway, Alan & Trott, James. *Design Patterns Explained: A New Perspective on Object-Oriented Design.*Addison Wesley, 2004.

Last update: 29 October 2018

TRAINING

Course Calendar (https://www.scaledagile.com/training/calendar/)

About Certification (https://www.scaledagile.com/certification/about-safe-certification/)

Become a Trainer (https://www.scaledagile.com/becoming-an-spc/)

CONTENT & TRADEMARKS

FAQs on how to use SAFe content and trademarks (https://support.scaledagile.com/s/topic/0TO0W000001YtQpWAK/get-permission-to-use-safe-content)

Permissions Form (http://www.scaledagile.com/permissions-form/)

Usage and Permissions (/usage-and-permissions/)

PARTNER

Becoming a Partner (http://www.scaledagile.com/become-a-partner/)

Partner Directory (https://www.scaledagile.com/find-a-partner/)

GET SOCIAL

Twitter (https://twitter.com/ScaledAgile)

Linkedin (https://www.linkedin.com/company/scaled-agile-inc-/)

YouTube (https://www.youtube.com/user/scaledagile)

SlideShare (http://www.slideshare.net/ScaledAgile)

RECENT POSTS

You asked for it: the What's New in SAFe 5.0 presentation is here (/blog/you-asked-for-it-the-whats-new-in-safe-5-0-presentation-is-here/)

Nov, 07th 2019

Enabling technical agility in the Lean enterprise vlog series: creating a shared understanding with Behavior-Driven Development (/blog/enabling-technical-agility-in-the-lean-enterprise-vlog-series-creating-a-shared-understanding-with-behavior-driven-development/)

Oct, 15th 2019

Introducing a preview of SAFe 5.0 (/blog/introducing-a-preview-of-safe-5-0/)

Oct, 02th 2019

SCALED AGILE, INC

CONTACT US

5400 Airport Blvd., Suite 300
Boulder, CO 80301 USA

BUSINESS HOURS

Weekdays: 9am to 5pm
Weekends: CLOSED

Privacy Policy (https://www.scaledagile.com/privacy-policy/)
Cookie Policy (https://www.scaledagile.com/cookie-policy/)