Organizing Domain Logic

Three primary patterns for organizing domain logic: *Transaction Script*, *Domain Model*, *Table Module*.

Transaction Script

Transaction Script takes input from the presentations, validates the input, stores the data in the DB, and invokes any operations from other systems.

PROS:

- Simple and most people understand it
- Works well with simple data source layer E.g. Row Data Gateway or Table Data Gateway.

Cons:

- Scales poorly
- Duplicate code when transactions need to do similar things

Domain Model

Domain Model is an object-oriented way of handling domain logic.

With domain model we build a model of the domain which is organized primarily around the nouns in the domain.

E.g. a leasing system would have classes for a lease, asset, etc.

The logic for handling validation would be placed within the model. E.g. a shipment object would contain logic to calculate the shipping charge for a delivery.

Pros:

- Domain logic is attached to the relevant object
- Easier to understand when approaching from an object oriented mindset.

Cons:

- Difficult to find logic if you're unsure what object you're dealing with.
- Data mapping becomes necessary when dealing with more complex logic between the domain and data layers
- Some engineers are unable to grasp the concepts.

Table Module

Table Module retrieves the necessary data at each step instead of having all the logic embedded on the object.

There are only instance of a contract in a system for any given contract. That means that to do work you must provide references (IDs and stuff) to the relevant information.

Considered the middle ground between Transaction Script and Domain Model.

GUIs typically rely on *Record Sets* and since Table Module also uses Record Sets it's easy to run a query, manipulate the data, run validation, and return the result to the GUI.

Pros:

- Easier to remove duplicate code
- Scales nicely for moderately complex systems

Cons:

- Unable to use many of the techniques that Domain Model uses at scale yet not as simple as Transaction Script
- Does not scale to highly complex domain.

Making a choice

Domain logic complexity determines which pattern you should use.

Simple - Transaction Script

Intermediate /w language support for RecordSets - Table Module

Complex - Domain Model

Other considerations are if the language provides native support for RecordSets. If there's support than Table Module becomes more attractive. (I don't really see the point of using anything other than Domain Model for anything relatively complex.)

You might use Transaction Script for some of the simple domain logic while using Domain Model for the rest.

Service Layer

One way to handle domain logic is to separate it into two. A **Service Layer** is placed over an underlying **Domain Model** or **Table Module**. **Transaction Script** is usually too simple to justify the separation of the layers.

With a service layer approach the presentation logic interacts with the domain purely through the Service Layer which acts as an API for the application.

Is a good place for transaction control and security.

A minimal implementation would be the service layer acting as a facade passing calls to lower level objects.

It still provides an API that's easier to use because it's oriented around use case.

Its also a convenient point to add transnational wrappers and security checks.