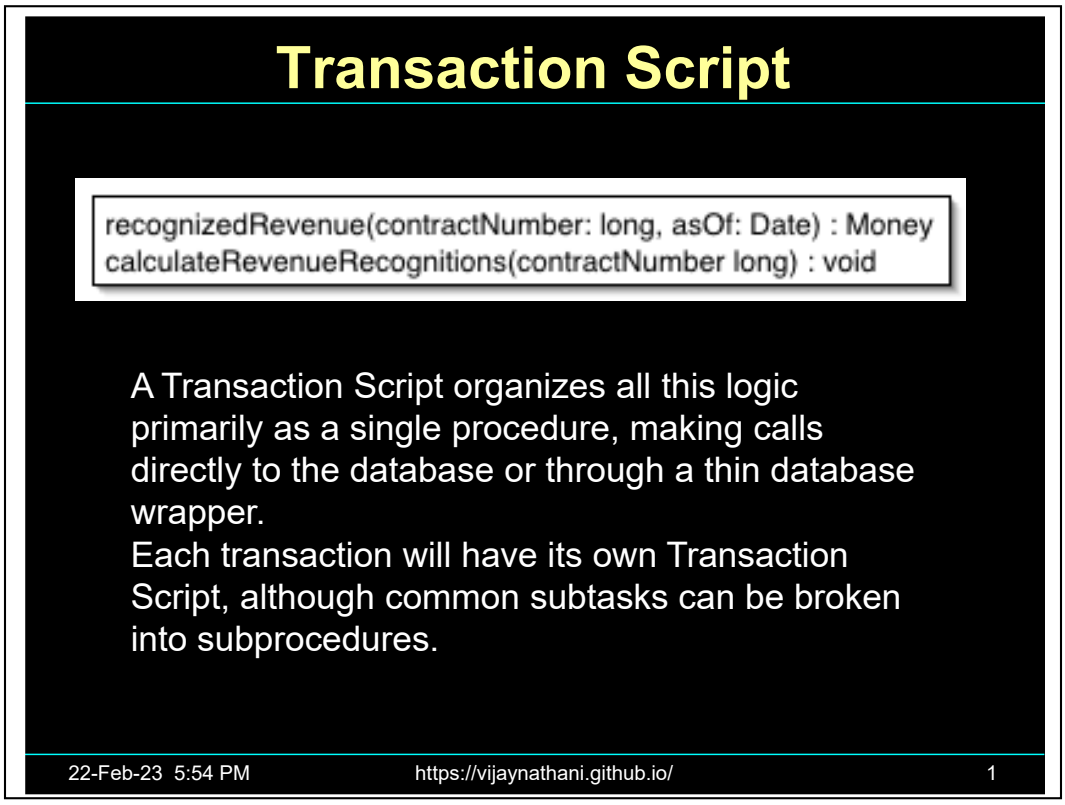
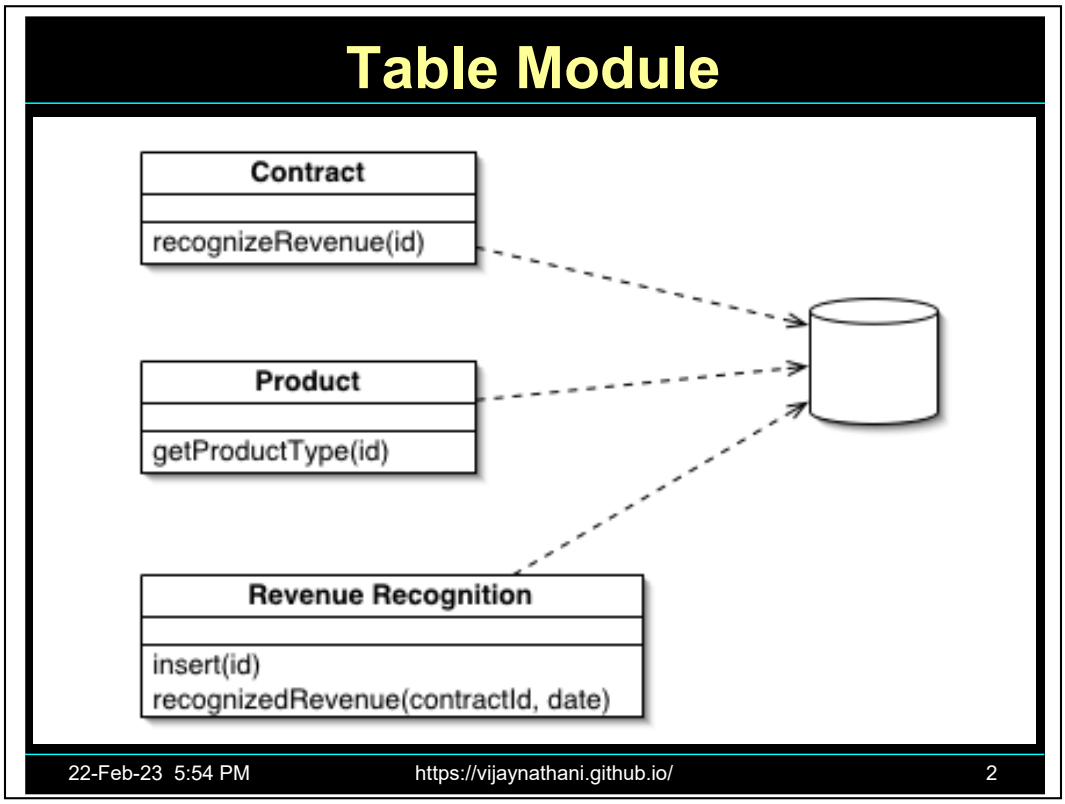


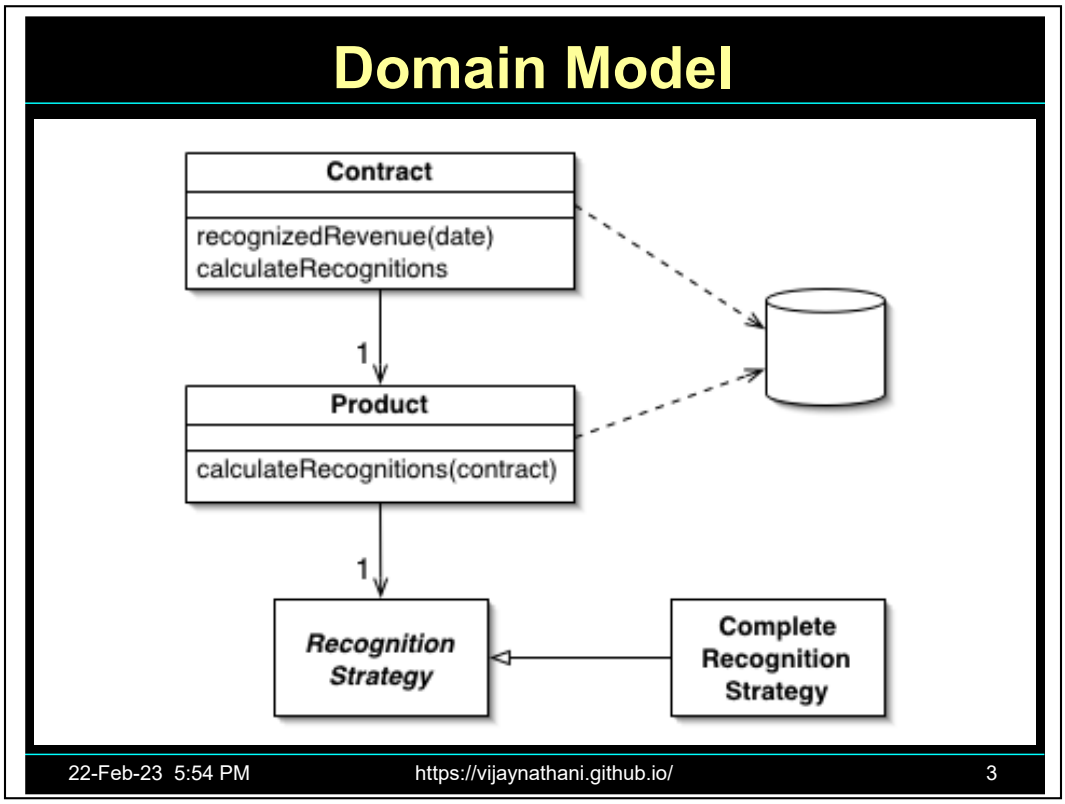
Transaction Script is similar to batch programs in that all functionality is described from start till end in a method. Transaction Script is very simple and useful for simple problems, but breaks down when used for dealing with high complexity. Duplication will be hard to avoid. Still, very many very large systems are built this way. Been there, done that, and I've seen the evidence of duplication even though we tried hard to reduce it. It crops up.

This model is useful for CRUD application and people coming from Data Processing (Cobol) background i.e. People with limit OO skills



A Table Module organizes domain logic with one class per table in the data-base, and a single instance of a class contains the various procedures that will act on the data. The primary distinction with Domain Model is that, if you have many orders, a Domain Model will have one order object per order while a Table Module will have one object to handle all orders.

Table Module encapsulates a Recordset, and then you call methods on the Table Module for getting information about that customer with id 42. To get the name, you call a method and send id 42 as a parameter. The Table Module uses a Recordset internally for answering the request. This certainly has its strengths, especially in environments when you have decent implementations for the Recordset pattern. One problem, though, is that it also has a tendency to introduce duplication between different Table Modules. Another drawback is that you typically can't use polymorphistic solutions to problems because the consumer won't see the object identities at all, only value-based identities. It's a bit like using the relational model instead of the object-oriented model.



A Domain Model creates a web of interconnected objects, where each object represents some meaningful individual, whether as large as a corporation or as small as a single line on an order form.

Domain Model instead uses object orientation for describing the model as close to the chosen abstractions of the domain as possible. It shines when dealing with complexity, both because it makes usage of the full power of object orientation possible and because it is easier to be true to the domain. Usage of the Domain Model pattern isn't without problems, of course. A typical one is the steep learning curve for being able to use it effectively.

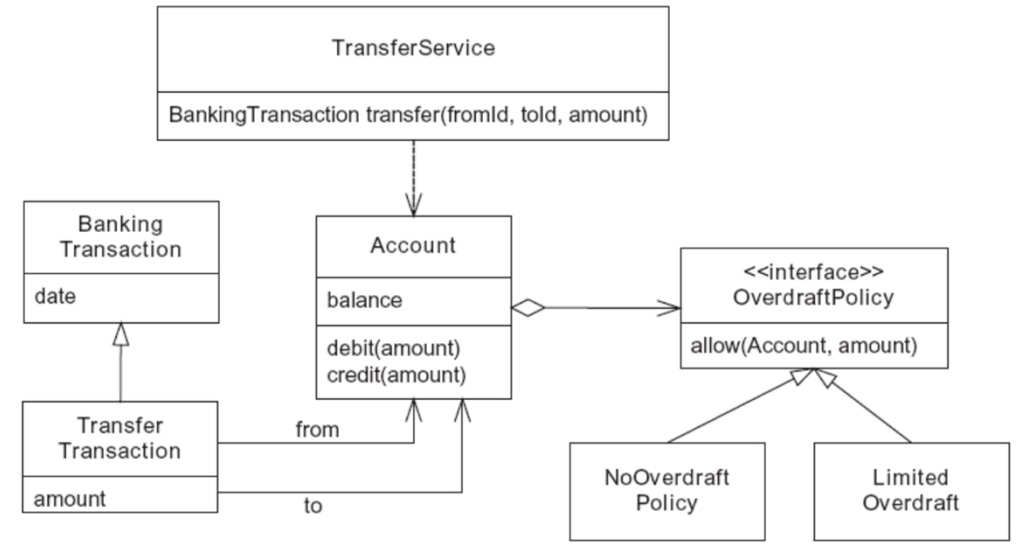
Domain Model is independent of whether we are using J2EE, Spring+Hiberate, .NET, Swing, C++ or windows rich client application.

Domain Model should enforce behavior and rules. It is usually developed in an iterative fashion.

Example UI

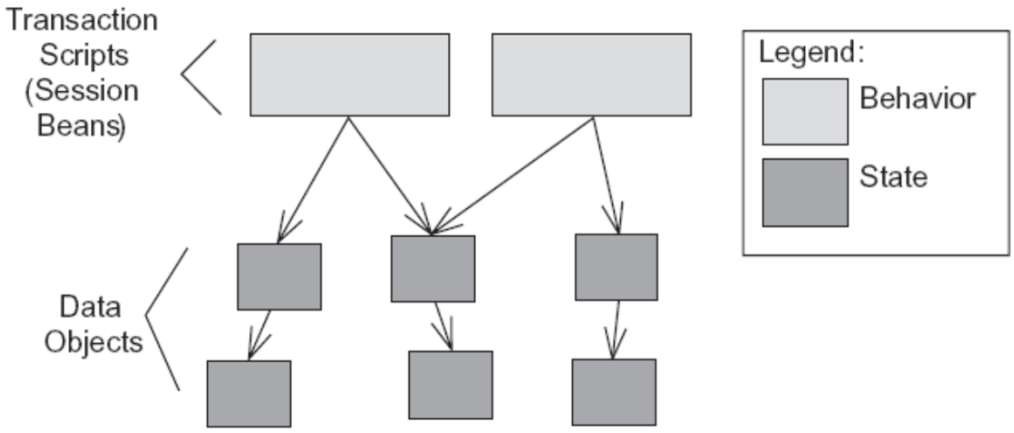
Example banking application

Domain Model



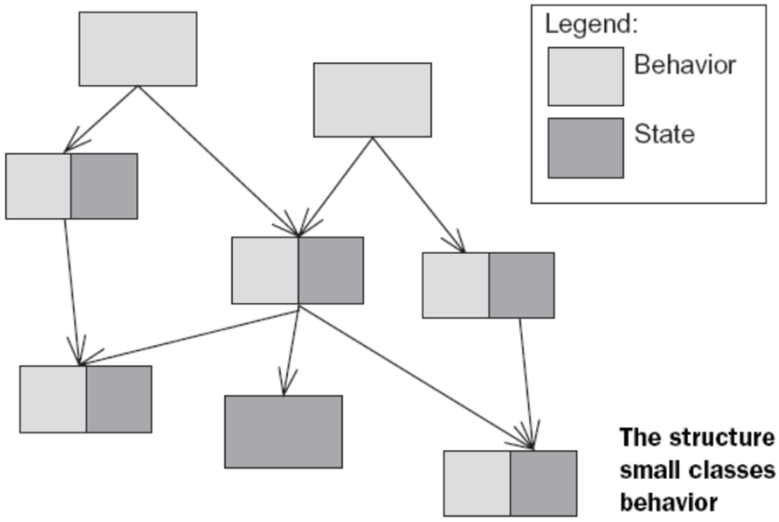
Transferring Money from one account to another

Transaction Script



The structure of procedural design: large transaction script classes and many small data objects

Domain Model



The structure of a domain model: small classes that have state and behavior

Helps in capturing domain expertise

Structures application code along the lines of business processes

- Through object-oriented design

Persistence is in second place

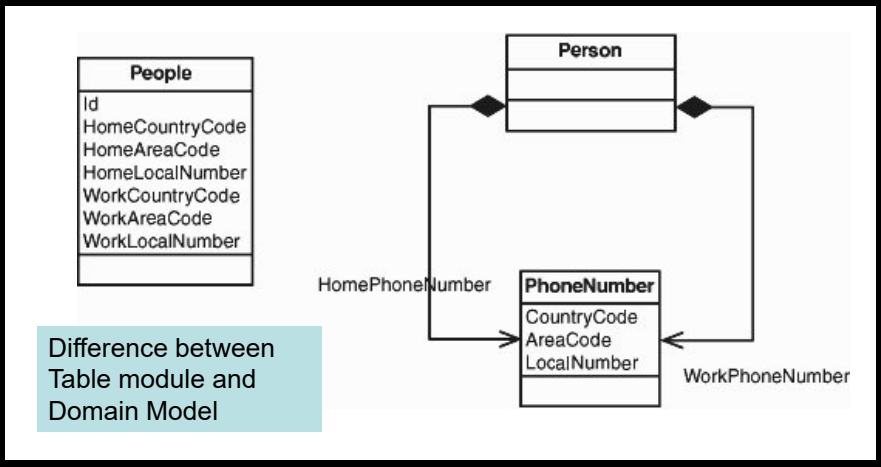
Can handle complexity

Need

- investment
- strong design and technical skills
- Regular access to domain experts

Disadvantages of Table Module

- Not cost effective
- No competitive advantage



Deciding whether or not to use the Domain Model pattern is actually an up-front architecture design decision, and a very important one because it will affect a lot of your upcoming work.

Domain Logic Patterns