Pragmatic Domain Driven Design (DDD) Patterns

Harihara Subramanian
Senior Principal, Software Architect

25/04/18
Core Domain is part of a software and ‘Raison D’etre’ of the Software
Travel
Banking
Problem Statement
Travel Business Excellence + Technology Excellence
Stakeholders

Business Analysts
Domain Experts
Product Owners

Software Architects
Developers

Domain Knowledge

Technology

Software Application
At Sabre, we practice

STRATEGIC PATTERNS

DESIGN PRINCIPLES

TACTICAL PATTERNS

EMERGING PATTERNS
Design Principles

01. Focus on what stakeholders want, when, and why

02. Gather requirements and capture required behaviors

03. Distill the problem space

04. Problem solver first, Technology second

05. Manage complexity with abstraction, create subdomains
Characteristics

Domain Model
Bounded Contexts
Aggregate
Aggregate Root
Scope

Strategic Patterns
- Ubiquitous Language
- Bubble Context
- Expose As a Service
- Domain, Subdomain, and Core domain

Tactical Patterns
- Modules
- Aggregate
- Domain Services
- Factories
- Repositories

Emerging Patterns
- Domain Events
- Event Sourcing
- Refactoring (towards deeper insight)
- Intent reveling interfaces
- RESTful API
- Micro Services
Strategic Patterns
Ubiquitous Language

- Ubiquitous Language

Strategic Patterns

Business Analysts
Domain Experts
Product Owners

Software Architects
Developers

Domain Knowledge

Technology

Software Application
# Ubiquitous Language Example

## Healthcare

<table>
<thead>
<tr>
<th>Universal / Ubiquitous Language</th>
<th>Pseudo Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>We administer Vaccines</td>
<td>AdministerVaccines()</td>
</tr>
<tr>
<td>We administer flu shots to patients</td>
<td>PatientsNeedaFluShot()</td>
</tr>
<tr>
<td>Nurse administer flu vaccines to patient in standard doses</td>
<td>Nurse -&gt; administerVaccine(patient, Vaccine.getStandardDose())</td>
</tr>
</tbody>
</table>

## Travel

<table>
<thead>
<tr>
<th>Universal / Ubiquitous Language</th>
<th>Pseudo Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>We book Air Tickets</td>
<td>ReserveAirTickets()</td>
</tr>
<tr>
<td>We reserve air tickets for business traveler</td>
<td>ReserverAirTicketsForbusinessTraveler()</td>
</tr>
<tr>
<td>Travel Agent book air tickets for business traveler by applying corporate travel policies</td>
<td>TravelAgent -&gt; reserveAir(getTraveler(Business), corporatePolicy(corp))</td>
</tr>
</tbody>
</table>
Bubble Context

- Best fit for
  - Iterative models
  - Legacy code involved
  - Need to apply DDD to legacy systems
- Provides clarity & directions
- Provides full control over the domain model
- No need of modifications to the legacy context
Expose As A Service

- (aka) Open Host Pattern
- Expose the Legacy System as a service
- Transaction Complexity
  - Mitigate with open Host API
- Legacy Context
  - Need Modification to the legacy context (unlike bubble context)
  - Standardization of consumable APIs may be challenging across different clients
Tactical Patterns
Domain Driven Design Tactical Patterns

![Diagram of Domain Driven Design](image-url)
Modules

Used in decomposing the domain model

Enables clarity in isolation

Quick understanding of domain models in code

Company
Team Responsible
Bounded Context
Domain
Modules

com.reservations.air.inventory.search
Aggregates

1. Decomposition of larger modules to smaller clusters
2. Technical complexities managed with high level of abstraction
3. Helps grouping of similar use cases and viewed as unified models
### <<ValueObject>> Profile ID
- Traveler ID

### <<AggregateRoot>> Trips
- DateOfTravel
- FindTripType()

#### <<ValueObject>> TripHistoryEntries
- TripType
- DateOfTravel
- Fare
- Origin
- Destination

#### <<ValueObject>> AccessTrips
- PNR
- DateOfTravel
- TripType

### <<ValueObject>> Past Trips
- Types
- DataRange

---

0... * connections between Profile ID and Trips, and Trips and Past Trips.
Emerging Patterns
Domain Events

Useful in triggering side effects

Used when events in one domain need to be cascaded to subdomains

Enforces consistency b/w multiple aggregates of same domain
Domain Events

Domain Model
(Ordering Micro Services)

Order Aggregate

Buyer Aggregate

Dominant Event

Order Started
Event Sourcing

- Provides simplification of events to persist and traverse
- Provide better performance
  - Events are immutable
  - Supports append only operations
- Useful during
  - A standalone object to access complex storage module
  - Audit Trails
  - Production Trouble shooting (store and replay)
- Seat allocation a good example
  - Find how many more seats available
  - Saved states involves bookings, cancellations, modifications etc.
When to choose Domain Drive Design

- Business requirements specific to domains
- Team
  - Never done such work before
  - Domain experts aligned with product vision
  - More complex business cases
- Design problem important to your business
Data points

01. Faster implementation to the product stack

02. Better product maintenance, Less defect leakages

03. Greater Collaboration with cross functional teams, Iterative friendly

04. Quick business value realization, early stakeholder involvement
References

• Domain Driven Design – Tackling complexity in the heart of software – Eric Evans (Pearson)
• Patterns, Principles, and Practices of Domain Driven Design – Scott Millet with Nick Tune (Wrox)
• Architectural Patterns – Pethuru Raj, Anupama Raman, Harihara Subramanian (Packt)
GREAT INDIAN DEVELOPER SUMMIT 2019
Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com  @greatindiandev  bit.ly/gidslinkedin