ESSENTIAL DOMAIN-DRIVEN DESIGN (DDD)

Participant Workbook

virtualgenius
leading by design

Table of Contents

Opener

Use a sharpie to write your first name in large letters on a nametag, and on the front of this workbook. Introduce yourself to someone from a different table group and share with each other what you already know about software design.

Return to your table group and share what you learned in your interviews. What common themes do you notice?

Domain Modeling – Case Study

Skim through the following story, and try to get a feel for the business domain based on what you learn.

The Dishwasher

Four years ago, on Oct 10, David and Claudia purchased a brand-new Whirlpool dishwasher. It was an expensive dishwasher, but a family with six children has a lot of dishes. They wanted a reliable, high-capacity dishwasher, and so they spent quite a bit of money on one that would do the job. The dishwasher had a retail price of $1,399 USD, but they were able to get it on sale for $1099 USD. It came with a 3-year manufacturer's warranty.

Our company, the *Elan Extended Warranty Company*, sends out a mailing just prior to the expiration of each customer's manufacturer's warranty. We keep those contact records in our Leads database, and we send each contact a letter explaining that for a small amount of money we can extend the warranty coverage over that product for another 3 years. David and Claudia thought, "Wow, we did spend a lot of money on this, and if we had to replace this right now, that's a big outlay for us. So let's pay a little extra for this warranty and have the peace of mind it will buy us." The pain of that initial purchase 3 years ago is in the background now, so that additional outlay for the extended warranty didn't seem as much to them.

The mailing went out on July 1. And on Oct 11, after responding to that mailing, David and Claudia became the proud owners of one of our company's extended warranties over their dishwasher. They became our customers.

On Nov 5, David called our call center and said the dishwasher is not working. What everyone fears actually happened, the moment the manufacturers warranty expired the product stopped working. The customer service representative (CSR) at the call center, John, opened a claim after he looked up David and Claudia's contract. He found their customer information, opened a new claim, and said, "What does the problem seem to be?" David said, "The problem is, as far as I can tell, that the soap dispenser is not opening during the wash cycle. No soap, dirty dishes. 6 children. Piles of dirty dishes."

John said, "No problem. I see this is a covered product under the extended warranty. The Nov 5 failure date of the product is after the Oct 11 effective date of the contract, and the Oct 10 expiration date is still several years away." So John creates a purchase order (PO), which is sent to a 3rd party-system managed by a servicer management company.

On Nov 12, Rick the dishwasher repairman comes out. David says, "The soap dispenser doesn't seem to be working." So Rick replaces the soap dispenser, runs the dishwasher and it's working.

Three days later, on Nov 15, David calls the call center again and talks with Jane the CSR. The dishwasher is still not working. David explains that the day after Rick left, the same problem occurred again, and has continued to happen. Jane looks up the contract and sees that the claim for David's contract has been closed because Rick had said it was fixed. So Jane reopens the claim, and creates a new PO.

Rick returns to David and Claudia's house on Dec 1st, and replaces the soap dispenser a second time. In the process of replacing the soap dispenser for a second time, he breaks one of the components inside

the dishwasher and has to replace that as well. The details for the components and their costs went on the second repair PO.

On Jan 15 Claudia has to make the call because David can't stand going through the process again. She says to Claire the CSR, "It's still not working. Help! Please don't send the same person. Rick was nice, but all he keeps doing is replacing the soap dispenser and the second time he replaced it he broke something else, so can you send someone different?" So this time we made sure a repairer was dispatched who was an expert in this particular brand and model of dishwasher. This repairman, Vince, was hard-core: he replaced the motherboard. He figured, based on running a diagnostic, that it was a software problem. He replaced the motherboard, and a couple of other products inside there, just in case. Before he left he ran the dishwasher and it worked just fine. That was on Jan 30. This has been going now since Nov 5, and this was David and Claudia's third repair.

On Feb 5th, David calls the call center again and speaks to Debbie the CSR and says, "I'm done." I don't want any more repairs on this product. Debbie says, "Wait. There's only been one claim, which is closed, but I see three Repair Purchase Orders on it. What is going on here? I see we're spending a lot of money on repairing this covered product, and it is approaching the point where more has been spent on the repairs than the product is actually worth - our contract is reaching our limit of liability."

Debbie says, "I'm going to reimburse you for the retail value of the dishwasher. I'll give you store credit for the market replacement value of the dishwasher so you can go purchase a new one from the store."

Debbie closed the claim, and terminated the contract, since it is fulfilled at this point (we've met all our legal obligations so there is nothing left to do on the contract). She emailed the Finance department to mail out the store credit.

So David and Claudia took their store credit and on March 10th purchased a new Whirlpool dishwasher with a three-year manufacturer warranty. The saga starts all over again, but at least now they have clean dishes again![1]

Review the story again, underlining anything you think might be important for understanding and modeling the business domain, and circling any words that might be key concepts for this domain.

[1] *This is a true story, though some details such as names and dates have been changed.*

Case Study – Review

1. How many PO's were created, and what kind of information did they contain?

2. What key steps were involved in closing the claim for the final time?

3. What are the top three domain concepts in the story related to closing the claim for the final time?

4. How does walking through a concrete business example like this help you understand the business domain?

5. How might understanding a concrete example like this help with automated acceptance testing? (hint: to function well as executable specifications, automated acceptance tests should use real business data)

What is a Model?

Based on what you learned in class, define in your own words what a model is:

What is the main criterion for determining the value of a particular model?

How does being able to make assertions about a model improve both shared understanding and the clarity of the model itself?

Why is it important to embrace the reality that there are always multiple models of the domain?

A. Each model can be targeted towards solving certain specific business problems most effectively
B. We want more models so we can write more software
C. It reduces overall complexity
D. We don't – there should only ever be one, true, model of the domain

Bounded Context

What are some clues to help you identify different bounded contexts?

Which of the following are accurate definitions of a bounded context?

A. A DBContext in Entity Framework
B. The conditions under which a particular model is defined and applicable
C. A subsystem boundary or business component
D. A Bounded Context is linguistic: a part of the system/project where language is consistent and rules agree
E. A description of a boundary (typically a subsystem, or the work of a particular team) within which a particular model is defined and applicable
F. The setting in which a word or statement appears that determines its meaning

Draw the initial context map for the case study below:

What negative things are likely to happen when the boundaries between two domain models aren't clear and well understood?

What is true about context maps?

A. Map the future!
B. They need to be right first-time
C. An accurate context map requires knowledge of what is actually happening in development
D. Ignore messy realities and make them as idealistic as possible
E. Context Map reflects what is. It is still left to us to make implementation decisions that make the best of it.

Practice drawing a quick initial context map diagram showing a few of the most important bounded contexts in your domain.

Reference Scenarios

How might reference scenarios help with onboarding new team members?

What is a complex/difficult important capability in your domain that might benefit from capturing as a reference scenario?

What are important attributes of reference scenarios?

A. Written up as a User Story, and prioritized as such in the Product Backlog
B. Describe an important business problem the software needs to solve
C. Provide shared understanding of business domain between domain experts and technical team
D. Documented in standard Use Case format
E. Harvested and documented for later reference in future modeling sessions
F. Concrete, realistic and specific

In Domain-Driven Design (DDD), what do we mean by "domain"?

A. A layer in the architecture where the business logic is kept
B. A sphere of knowledge, influence, or activity (the subject area to which the user applies a program)
C. A model
D. The first word in the name of the book title

Ubiquitous Language

Which of the following is the definition of Ubiquitous Language given in class?

A. A handful of classes and patterns that shape the core flow of the system being built
B. A language structured around the domain model and used by all team members within a bounded context to connect all the activities of the team with the software
C. The logical architecture of the system
D. An enterprise data model, forged by Sauron in the fires of Mordor

Summarize, in your own words, your understanding of "ubiquitous language" in DDD.

What are the benefits of growing a ubiquitous language for your project?

Where are places in your development work where you could benefit from growing a robust ubiquitous language?

Which of the following are true about Ubiquitous Language?

A. The team should work toward a language that enables natural, precise statements about the model and domain
B. Increases the amount of translation for team members. Translation is good!
C. Cultivates a language of domain terms, not tech terms
D. All team members should exercise the ubiquitous language - a lot
E. Increases the distance between domain experts, developers, requirements and code
F. A change in the ubiquitous language is a change in the model
G. Experimentation with language is critical to developing a truly ubiquitous language
H. Provides a higher-level language to describe and discuss service contracts in speech or writing

MY NOTES

Design and Agile

How central is good design to your process in your team?

Review the front of the User Stories Reference Guide and highlight anything you find interesting. Once you're done, share with your table group what you highlighted, and any questions it raised for you.

What is one way you might better integrate design into your team process?

What is an example from your own project work for each of the conjunction slicing patterns?

Operations -

Data Variation –

Data Entry Methods –

Major Effort –

Workflow Steps –

How might adopting Rule Relaxation and Design Probes as story splitting techniques improve a team's productivity?

What is an example of something your team will need to deliver that could benefit from one of the Rule Relaxation or Design Probe approaches?

Three Pillars of DDD

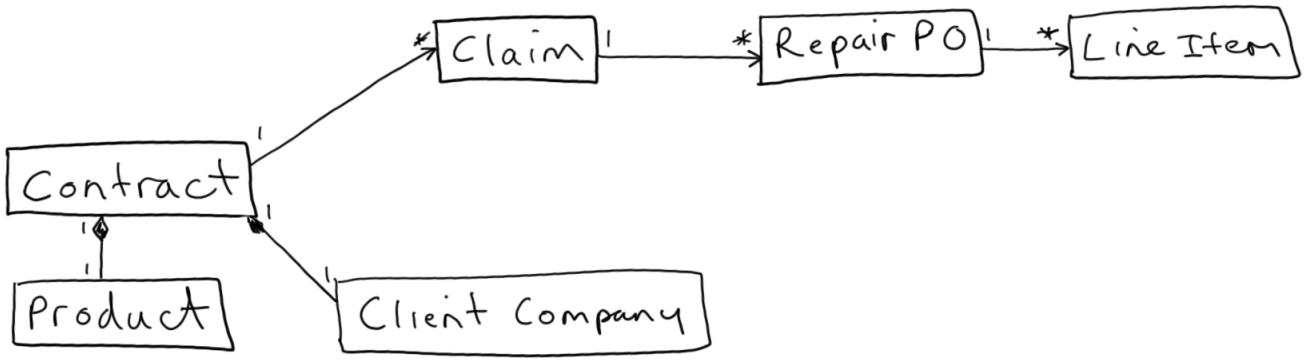
Draw a diagram below of the three pillars of DDD

Which of the three areas intrigues you most?

Case Study – Domain Model

Version 1.0

Here is a class diagram for the part of the domain model relevant to our case study.



What questions does this raise for you about the various concepts represented in the diagram?

Draw an instance diagram on a flipchart that reflects the state of the system at the end of the case study. Include any specific details that might be relevant to talking with a domain expert about tracking progress towards the limit of liability.

How did diagramming a specific example together using an instance diagram help:

- surface hidden assumptions?

- clarify the domain?

- drive shared understanding?

What's an example from your team domain that could benefit from this model exploration technique?

When collaborating with domain experts, what are important things to keep in mind?

A. Cultivate a strong relationship between the domain experts and development team
B. Try to get to the requirements as quickly as possible and ignore everything else domain experts say as it's probably boring and irrelevant
C. Restrict the conversation to only what the software has to do
D. Focus on the ubiquitous language
E. Once you can get the right answer, you are done. Yes!
F. Listen for clues
G. You must literally *become* a domain expert if you want to succeed with DDD
H. Be open to shifts in assumptions that break your model
I. Keep definitions as vague and abstract as possible
J. Ask for recommended reading
K. Build a relationship of mutual respect
L. Avoid walkthroughs, they will only distract you from modeling
M. Learn enough to have an intelligent conversation

MY NOTES

Domain Distillation

Draw the Purpose Alignment Model and how it maps to DDD strategic domain distillation below.

Classify each of the following practices according to which single subdomain they might *most* be appropriate

A. Outsource development to an offshore team [Core | Generic | Supporting]
B. Buy an off-the-shelf software package [Core | Generic | Supporting]
C. Do model exploration in collaboration with domain experts [Core | Generic | Supporting]
D. Do lots of experiments and make many mistakes to maximize learning [Core | Generic | Supporting]
E. *Never* buy an off-the-shelf software package [Core | Generic | Supporting]
F. Use an open source framework [Core | Generic | Supporting]
G. Utilize the skills of your most experienced developers and skilled domain modelers [Core | Generic | Supporting]
H. Collaborate intensively with the domain experts [Core | Generic | Supporting]
I. Bring in a team of contractors [Core | Generic | Supporting]
J. Avoid churn in the staff to encourage deep domain knowledge acquisition over a period of time [Core | Generic | Supporting]
K. Hire an intern to do the work [Core | Generic | Supporting]
L. Isolate in a clean, explicitly bounded context [Core | Generic | Supporting]

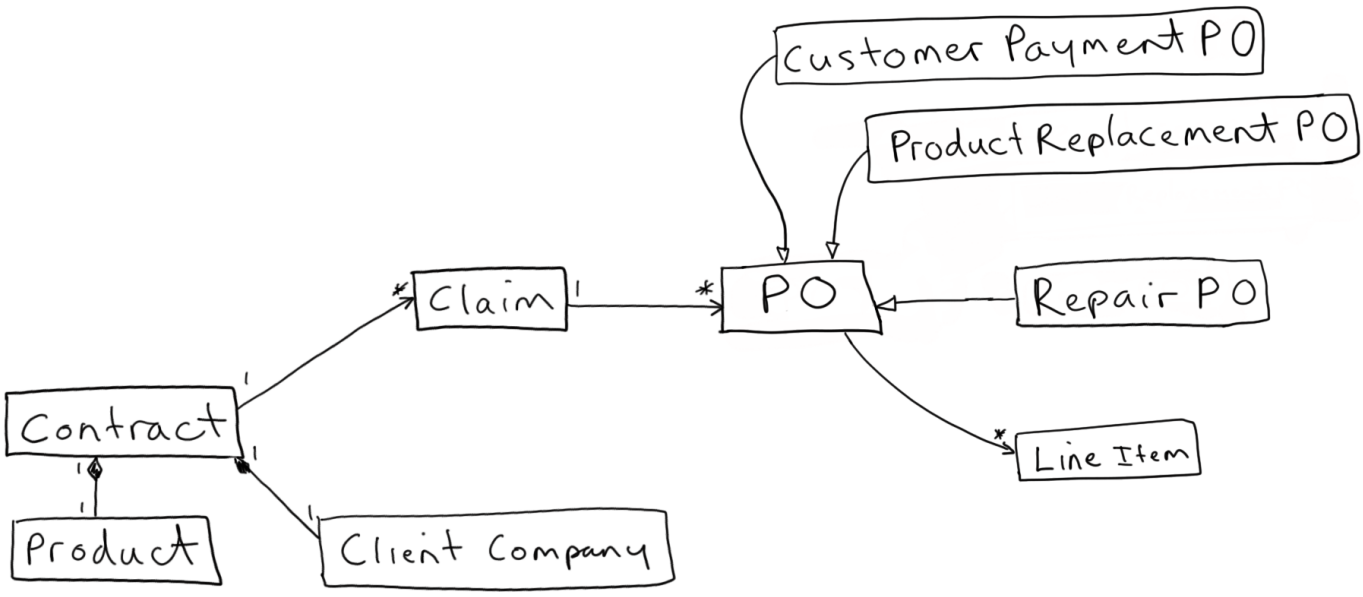
Which of the following statements about core domain are true?

A. Core domain is primarily about core features
B. Market differentiation is a key component to focus on when trying to identify your core domain
C. There is always only one core domain at any one time
D. Core domain is primarily about core business
E. Spend the effort in the core to find a deep model and develop a supple design
F. The core domain never changes
G. Keep the core domain small
H. Apply top talent to the core domain, and recruit accordingly
I. Justify investment in any other part by how it supports the distilled core

How clear are you on what is generic, supporting, or core domain for your own project work?

Breaking the Model

The system now needs to handle issuing store credits for customers, and supporting product replacements. The software architect talked briefly with the product owner about what is needed, and came up with the following diagram as a proposal for an updated domain model.



Compare and contrast the current (v1.0) domain model with the proposed model. How well does the proposed model reflect the ubiquitous language?

Does the model have any friction points in handling the new product replacement scenario properly?

On a flipchart, come up with 3 different diagrams, each representing a domain model with slightly different language and modeling choices.

Cultivating Collaboration

What makes open-ended questions better than binary questions for gaining domain understanding?

How does presenting multiple options when doing language and model exploration cultivate collaboration?

Capture the list of open-ended questions we talked about in class

Which question above do you plan to experiment with using to cultivate more collaboration and shared understanding?

Model Exploration Whirlpool

Draw the model exploration whirlpool below

Which of the following are true statements about the Model Exploration Whirlpool?

A. The team must complete the Whirlpool at the start of a project, as part of the Design Phase
B. The Whirlpool shifts modeling from "push" to "pull"
C. The team can only jump into the Whirlpool once they have a Reference Scenario
D. Only developers are involved in the Whirlpool
E. The Whirlpool never involves writing actual code – why waste time doing *that*?
F. Brainstorming sessions are an integral part of the Whirlpool
G. The Whirlpool cultivates the creative collaboration of the technical team and their business counterparts
H. Selecting the right scenarios and describing them properly is critical to successful modeling

Which of the following are indicators that it could be time for modeling?

A. Communications with stakeholders deteriorate
B. That phase has been reached in the software development lifecycle (SDLC)
C. Solutions seem more complex than the problems
D. Velocity slows (completed work becomes a burden)
E. Management says so
F. The software will be released to production tomorrow
G. The team is moving into a new domain area for the first time and needs a minimal shared view to get started

MY NOTES

The Problem of Vague Intentions

Working in pairs in a text editor, IDE, or in your browser, look through the code in the various files.

Where is the business logic/behavior?

What does it do?

What makes it difficult to understand?

What questions does it raise for you?

What do you notice that is ambiguous and confusing in the unit tests?

The Problem with Primitive Obsession

Look at the attributes/properties for the Contract class. What different responsibilities /concepts seem to be represented/expressed by the various primitives (e.g. int, double, decimal, Date, etc)?

What questions do you have about what they mean?

How does having many primitives increase the "accidental complexity" (i.e. technical complexity) of an object?

What are ways the excessive use of primitives obscures/hides important domain concepts?

The Problem of Temporal Coupling

Look at the date properties for Contract (e.g. EffectiveDate, ExpirationDate, etc.), which are essential attributes of Contract. Based on the constructor for Contract, what potential issues/bugs do you see arising with setting these values for a Contract?

Temporal coupling is a design smell that happens between sequential method calls when they must stay in a particular order. How is this a form of temporal coupling?

Making the Implicit Explicit

Go to the refactoring branch.

Look at the new limitOfLiability() method in Contract. How has adding this limitOfLiability() method improved the maintainability and readability of the code?

What is the connection between adding this method and growing our ubiquitous language?

How has this improved our domain model?

How did we separate the concept of Limit of Liability from the Contract lifecycle?

Introducing a Value Object

Look at the new TermsAndConditions object we've added as part of the Contract.

A *value object* is an object that *measures, quantifies, or describes something*. Which of the following did we need to do to make TermsAndConditions a value object?

A. Remove all setters
B. Make functions/methods side-effect-free
C. Defensive copying of parameters - not passing in or handing out objects by reference
D. Implement hash code and equals for all properties
E. Give it an identity field
F. Pass all dependencies in through the constructor

What do you notice about the unit tests for TermsAndConditions?

How has grouping primitives into TermsAndConditions as a value object within Contract improved overall:

- testability?
- reducing temporal coupling?
- thread safety?
- code maintenance?

What does the annuallyExtended() method do, and how is it used by the Contract entity?

What other logic does Contract now delegate to TermsAndConditions?

How does incorporating this new value object simplify the Contract entity and focus it more tightly on identity and lifecycle?

"Classes should be immutable unless there's a very good reason to make them mutable….If a class cannot be made immutable, limit its mutability as much as possible."

— Joshua Bloch
Effective Java

Value Objects and Entities

Capture the concept-maps about value objects and entities below

Why should we be biased towards modeling concepts as value objects?

What did the instructor mean by making value objects centers of gravity for behavior?

When logic is complex, look for implicit concepts. Where are likely places to find clues for implicit concepts?

A. In the way spoken statements are phrased.
B. Ambiguous and confusing unit tests
C. In the awkwardness of particular parts of the design
D. Astrological charts
E. Within code obscured by many implementation details or complicated compensation logic

A good model is one that you can make assertions about. What else is important about assertions?

A. Only bullies make assertions
B. If assertions cannot be coded directly in your programming language, write automated unit tests for them
C. Assertions encourage the situation where the only way to understand a program is to trace execution through branching paths
D. Assertions define contracts of services and entity modifiers
E. Avoid expressing assertions in the ubiquitous language
F. Write assertions into documentation or diagrams where it fits the style of the project's development process

Which of the following statements are true about entities?

A. Have a thread of identity over time
B. Are immutable
C. Focus their definition on identity and lifecycle
D. Fill them up with as many properties as possible
E. Exist in the real world
F. Cannot be Aggregate roots
G. We should try to model most concepts as entities

Which of the following statements are true about value objects?

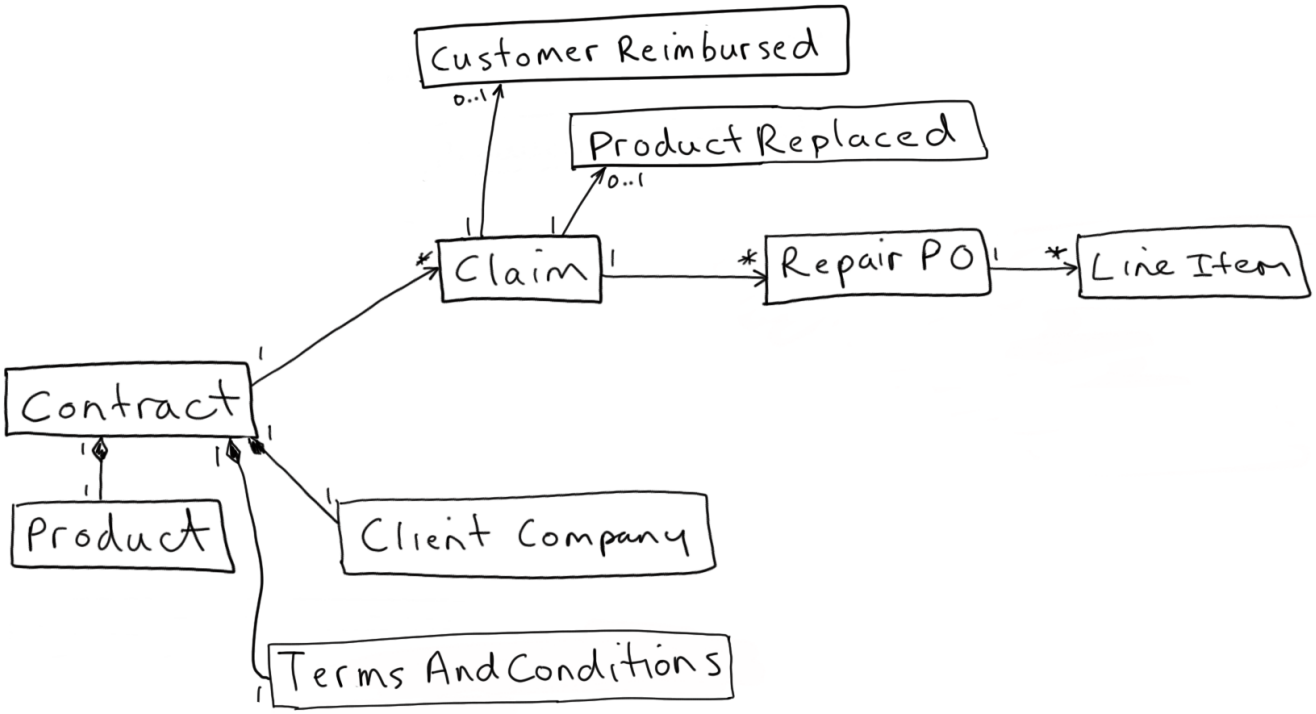
A. Exist in the real world (eg. a dollar bill is always going to be a value object)
B. Have a thread of identity over time
C. Are immutable
D. Are equal when all their properties are equal
E. Cannot have behavior
F. Represent things, rather than measuring, quantifying, or describing something
G. We should be biased towards modeling concepts as value objects
H. Can be aggregate roots, or whatever they want to be. They are in charge of their own destiny and nothing can stand in their way!
I. Must have side-effect-free functions
J. Improve encapsulation

MY NOTES

Domain Events

Write down at least three key things to remember for domain events

Case Study – Proposed Model (with Domain Events)



What improvements do you notice in the ubiquitous language in the proposed model above?

How does using domain events in the model above make it a more useful, expressive and robust model?

Aggregates

What are some specific examples of concurrency, transactional and distribution challenges you see in your own systems?

What are the five rules of thumb for aggregate design?

1.
2.
3.
4.
5.

What are some important things to remember about aggregate roots?

Which are the following are true statements about aggregates?

A. Data consistency rules apply within aggregates, so each aggregate is its own "island" of data consistency
B. Aggregates can be embedded within each other
C. Within an aggregate, invariants apply at every transaction commit
D. The aggregate root must be a value object
E. The aggregate root must be an entity
F. An aggregate is a cluster of domain objects that can be treated as a single unit
G. Aggregates enable data consistency rules to be represented within the domain model
H. Aggregates are a waste of time, because everything should be connected to everything else. Barrel of Monkeys!

When making changes across aggregate boundaries, what should we keep in mind?

A. Aggregates are "eventually consistent" with each other
B. Aggregates cannot ever be shared between bounded contexts, except in a shared kernel
C. All aggregates must always be kept consistent with each other
D. Aggregate-related updates propagate asynchronously through the system
E. Domain events are the only correct way to handle cross-aggregate updates
F. We are ok with data being stale between aggregates, at least for a period of time

Eventual Consistency

What is likely to happen if we try to make too many things logically consistent rather than embrace some latency?

How does embracing eventual consistency across aggregates improve your ability to scale your system?

Sketch an example of some concept in your domain that should be modeled as an aggregate

What are some practical ways to deal with data latency from a UI perspective?

Hexagonal Architecture

Draw a diagram describing hexagonal architecture

What does the hexagonal architecture approach emphasize compare to a layered architecture approach?

What might be possible challenges with implementing a hexagonal architecture?

How does hexagonal architecture positively impact the testability of a system? What are the other key benefits?

Services

In DDD, what is the difference between a domain service and an application service?

What will happen to our domain model if we don't make an ongoing effort to embed the ubiquitous language in our domain services?

How do domain services help isolate infrastructure concerns from our domain model?

How does abstracting verbs/actions into domain services improve testability? Maintainability?

Repositories

What is a repository?

When do aggregate-oriented databases work best?

When are aggregate-ignorant databases a better choice?

What is true about the relationship between repositories and aggregates?

A. Query access to aggregates is provided via repositories
B. Repositories are services that provide the illusion of an in-memory collection of all objects of that aggregate's root type
C. Aggregates can only be persisted through repositories that save to relational databases using an object-relational mapper
D. *What* relationship? Aggregates and repositories have nothing to do with each other.
E. Repositories must inherit from a base class that provides CRUD (Create-Read-Update-Delete) functionality
F. Repositories must always be suffixed with the word "Repository" (for example, CargoRepository)
G. Repositories are used to create aggregates for the very first time (i.e. they do not yet exist in the system)

MY NOTES

Context Mapping

Draw summary diagrams of the various types of context relationships below

What are potential benefits of context mapping?

A. Mapping the future configuration allows you to create an enterprise modeling strategy
B. Context maps are a big-picture complement to core domain, helping you see the terrain and understand how well you are set up to do core domain work
C. It helps you ask the right questions as you scale development across multiple teams and contexts
D. The process of mapping contexts makes the state of context boundaries more clear
E. The resulting context map will help you see where context boundaries are not aligned with subsystem boundaries
F. Context mapping is a helpful technique for non-technical people to see the technical and organizational landscape

MY NOTES

What are the most important things you want to remember for the four strategies?

According to the instructor, how is introducing a difficult new set of development principles and techniques best done?

MY NOTES

Event Sourcing

Summarize your key insights on event sourcing below

MY NOTES

MY NOTES

Learning Log – Day 1

Here is your place to capture your "aha's" from today. Include such things as:

- things you learned,
- new insights you want to remember,
- cool things you heard other people say.

What do you plan to experiment with in the next week or so based on what you experienced today?

MY NOTES

Learning Log – Day 2

Draw a diagram that captures and connects the ideas from your biggest "aha's" from today.

What technique(s) are you most looking forward to experimenting with? How are you going to apply it in your work?

Learning Log – Day 3

Here is your place to capture your "aha's" from today, and connect them with what you learned on previous days. Include such things as:

- things you learned,
- new insights you want to remember,
- cool things you heard other people say, and
- things you want to experiment with based on what you experienced.

Resources

Here's a list of further resources to get you further in your DDD journey:

Patterns, Principles and Practices of DDD - Scott Millett and Nick Tune

> .NET-focused, but highly recommended. *This is the DDD book I recommend you start with.*

EventStorming - Alberto Brandolini

> On Leanpub in electronic form. Still a work-in-progress, but has all the essential information on getting started with EventStorming

Domain-Driven Design - Eric Evans

> The classic, definitive book on DDD (hint: read parts in this order: 1, 4, 2, 3)

Implementing Domain-Driven Design (IDDD) - Vaughn Vernon

> Java-focused, heavy on technical approaches. Good chapter on aggregate design.

The Anatomy of DDD - Scott Millett

> Fantastic summary infographic (pdf). This was provided to you as part of the pre-class warmup.

Functional and Reactive Domain Modeling – Debasish Gosh

> DDD in a functional style, with examples in Scala.

Versioning in an Event Sourced System - Greg Young'

> Greg's new book, hosted on Leanpub, about how to manage change in an event sourced system.

Gamestorming - Dave Gray, Sunni Brown, James Macanufo

> If you want to facilitate EventStorming sessions effectively, this is a great book for tips on facilitation and collaboration.

DDD Community site (dddcommunity.org)

> Somewhat dated, but plenty of great material

virtualgenius.com/resources

> My talks, blog posts, podcasts, etc.

About Virtual Genius LLC

Virtual Genius LLC helps software teams succeed, with customer-focused collaborative design at the center of everything the team does. This approach emphasizes principles and practices from design thinking, domain-driven design (DDD), user-experience design (UXD), and behaviour-driven development (BDD).

We provide world-class training and coaching (both onsite and remote) in BDD and DDD. Contact us if you want to take your team's skills to the next level.

Contacting Paul

Email: paul@virtualgenius.com
Twitter: @thepaulrayner
Blog: thepaulrayner.com
Website: www.virtualgenius.com