STRATEGIC

DOMAIN-DRIVEN DESIGN

Nick Tune - @ntcoding
Hey Nick, I just stumbled across one of your videos and must say I'm impressed. You are like a combination of Martin Fowler and Eminem :)

— Anon
THE 3 VIRTUES OF MODULARITY

1. ☔️ Isolation
2. ???
3. ???
MILLER’S LAW

The number of objects an average human can hold in working memory is \(7 \pm 2\)

https://en.wikipedia.org/wiki/The_Magical_Number_Seven,_Plus_or_Minus_Two
THE 3 VIRTUES OF MODULARITY

1. 🌧️ Isolation
2. 🧠 Comprehensibility
3. ???
THE 3 VIRTUES OF MODULARITY

1. ☔️ Isolation
2. 🧠 Comprehensibility
3. ✈️ Parallelisation / Autonomy
"If we achieve a loosely-coupled, well-encapsulated architecture with an organizational structure to match, we can achieve better delivery performance... and substantially grow the size of the engineering organization and increase productivity linearly."
[In our study at Thoughtworks we found] work takes an order of magnitude longer when it leaves a team.

— James Lewis (@boicy)
Finding service boundaries is really damn hard... there is no flowchart

— Udi Dahan (@UdiDahan)
Goodbye Microservices: From 100s of problem children to 1 superstar

Alexandra Noonan on July 10th 2018

https://segment.com/blog/goodbye-microservices/
STRATEGIC CHALLENGES

• Local vs global complexity
• Coupling & bottlenecks between teams
• Political disputes
• Inconsistent UX
• Fragile technical integrations
Strategic DDD: designing sociotechnical systems according to the 3 virtues of modularity.

** This is my interpretation not an official definition
DOMAIN-DRIVEN DESIGN: THE ELEVATOR PITCH
Businesses need to sustainably deliver value, but software grows less maintainable over time.
To create evolvable software we must be committed to continually reducing complexity.
To minimise complexity, we need to understand what complexity is truly essential in order to model our domain.
initial models usually are naive and superficial, based on shallow knowledge.

— Eric Evans (@ericevans0)
The refactorings that have the greatest impact on the viability of the system are those motivated by new insights into the domain.

— Eric Evans (@ericevans0)
KEY DDD PRINCIPLES

• Everyone speaks the business language
• Whole team constantly strives to improve domain knowledge
• Optimise for core business domains (where competitive advantage can be gained)
• Make implicit business concepts explicit in code model
Implicit domain concepts

```java
public boolean isApplicable(Customer customer) {
    return customer.loyaltyPoints() > 5000;
}
```

Explicit domain concepts

```java
public boolean isApplicable(Customer customer) {
    return isGoldCustomer(customer) ||
        isPlatinumCustomer(customer);
}
```
4 ELEMENTS OF DDD

- Domain Discovery
- Tactical Modelling
- Refactoring to Deeper Insight
- Strategic Design
DOMAIN DISCOVERY
Strategic DDD is not ivory tower architecture. It requires continuous domain discovery.
He tried **UML** with domain experts!
MODEL DOMAIN AS TIMELINE

- CFP Opened
- Talk proposal submitted
- CFP Closed
- Talk Accepted
EXTERNAL SYSTEMS, MONEY, UX

- Ticket sold
- Tickets downloaded
- Speaker dropped out
- Event booking service

💰
COMMANDS, USERS, POLICIES

- **Submit Proposal**
- **Talk proposal submitted**
- **Proposal Limit Policy**
- **Talk Proposal Rejected**
PERSONAL ES TRIUMPHS

• Uniting my teams and ‘the business’
• Modelling entire domains
• Engaging many domain experts for hours
• Gaining deeper understanding of domains
LEARN EVENTSTORMING

leanpub.com/introducing_eventstorming

eventstorming.com

github.com/mariuszgil/awesome-eventstorming
WHAT'S WRONG?

public class Product {
    public void allocate() {...}
    public Locations stock() {...}
    public Recommendations similar() {...}
    public Price priceFor(CustomerType..) {...}
    public PurchaseOrder buyFrom(Supplier..) {...}
    public Boolean canShipTo (Country..) {...}
}
CONTEXT
DEPENDENCIES PROLIFERATE AS THE CODEBASE GROWS
Dependencies increase coupling making code unintelligible, fragile, and harder to change.
Large models are conducive to **unnecessary** coupling.

So much coupling is completely unnecessary!
How can we make it easy to add necessary coupling while making it harder to add unnecessary coupling?
Multiple models are in play on any large project... Explicitly define the context within which a model applies.

— Eric Evans (@ericevans0)
public class Product {
    public void allocate() {...}
    public Locations stock() {...}
    public Recommendations similar() {...}
    public Price priceFor(CustomerType..) {...}
    public PurchaseOrder buyFrom(Supplier..) {...}
    public Boolean canShipTo (Country..) {...}
}
Just because a thing has a physical representation, it doesn’t mean it needs a single representation in code.
COUPLING INCREASES COMPLEXITY

Inventory

Personalisation

Fulfilment

Pricing

Product
COUPLING DANGERS (page 1 of many)

- Changes rippling across the codebase
- One team ‘breaking’ another team’s code
- Frequent coordination & merge conflicts
- Generic model that loses domain focus
- Lack of ownership and accountability
Each product model applies within a separate context, facilitating isolation, comprehensibility and autonomy.
SAME WORD DIFFERENT MEANING

Sales

Ticket
- Number of seats
- Show length
- Cost
- Location

Customer Support

Ticket
- Severity
- Date raised
- Category
Generally speaking, there is a correspondence of one team per BOUNDED CONTEXT.

— Eric Evans (@ericevans0)
AUTONOMOUS TEAMS

- Fulfilment: Product
- Inventory: Product
- Pricing: Product
- Shipping: Product
There are an unlimited variety of situations and an unlimited number of options for drawing the boundaries of BOUNDED CONTEXTS.

— Eric Evans (@ericevans0)
Look for heuristics that challenge your assumption or disprove your theory... that’s when you learn something

— Mathias Verraes (@mathiasverraes)
MODELLING PROCESS

1. Explore contexts on EventStorm
2. Sketch out multiple models
3. Challenge models with design heuristics & use cases
CASE STUDY: GOVERNMENT TAX

Review

Resubmit

Renegotiate

Case Mgmt.

UI

API

UI

API

UI

API

UI

API

ntcoding
DESIGN HEURISTIC
FIND CONTEXTUAL LANGUAGE

Segregate parts of the domain where specific words or phrases apply. This allows us to create explicit, comprehensible models.
DESIGN HEURISTIC
ALIGN BY DOMAIN EXPERT

Different domain experts will care about different parts of the system indicating inherent domain cohesion.
DESIGN HEURISTIC

ALIGN WITH CO-CHANGE

Put boundaries around things that change together because it’s easier to make changes within a boundary than across boundaries. Aka minimise dependencies.
HIGH CO-CHANGE -> BOTTLENECK

Resubmit
Renegotiate

Case Mgmt.

Requires corresponding change
FORENSIC ANALYSIS

Analyse source control history to identify dependencies in sociotechnical systems
DESIGN HEURISTIC
ALIGN TO THE ORGANISATION

Create boundaries that reflect the structure of the organisation so that teams own everything they need to run their part of the business.
CONTEXTS VS EXPERIENCES

Review
- UI
- API
- Admin UI

Resubmit
- UI
- API

Renegotiate
- UI
- API

Case Management UI
ALIGN WITH BIZ VALUE

Isolate parts of a system based on their value to the business enabling a fine-grained investment strategy and optimal ROI.
INVESTMENT FLEXIBILITY
**CASE STUDY: ADTECH CONTEXTS**

95% of revenue is generated by customers running ad campaigns on Facebook.

Monolithic codebase and shared DB optimised for reuse
DDD SUBDOMAIN JARGON

Core
A part of the domain where ROI and/or differentiation potential is high.

Supporting
Specific to current domain and needed in order to deliver core domains but little opportunity for differentiation.

Generic
A capability that is not unique to the current domain and holds little potential for differentiation. Consider buying.
WHAT IS CORE CAN CHANGE

Slack started life as an internal chat system. It is now the core domain worth $4bn.
Caveat: Aligning purely by value can result in a lot of dependencies.
Different parts of a system can relate to each other across multiple dimensions. It’s not a case of simply classifying related concepts.
CASE STUDY: FINANCIAL PRODUCTS

- High Value Credit Cards
- High Value Mortgages
- High Value Loans
- Medium Value Credit Cards
- Medium Value Mortgages
- Medium Value Loans
- Low Value Credit Cards
- Low Value Mortgages
- Low Value Loans

💰💰💰
💰💰💰
💰💰💰
💰💰💰
💰💰💰
💰💰💰
DESIGN HEURISTIC

ALIGN BY PRODUCT TYPE

Put boundaries around each category of product/service your company sells because those concepts will be highly cohesive.
DESIGN HEURISTIC

ALIGN WITH USERS

Align contexts with users so each type of user has 1 team who are fully committed to giving that user the best possible experience.
Understand your business model to identify what is valuable to your company.
1. Understand the business model
2. Explore contexts on EventStorm
3. Sketch out multiple models
4. Challenge models with design heuristics & use cases
Understand the patterns in your domain to determine which heuristics you should trust.
ENTITY LIFECYCLE PIPELINE

Review ➔ Resubmit ➔ Renegotiate
CASE STUDY: AIR TRAVEL PLANNING

Phase 1 user (~ 6 months)
- Routing

Phase 2 user (~3 months)
- Fleet Allocation

Phase 3 user (~1 months)
- Crew Selection

Rules
PROPOSAL PIPELINE

Routing → Fleet → Crew
OCTOPUS CONTEXT

GDPR

ntcoding
OCTOPUS CHALLENGES

• Try to centralise complexity in the Octopus
• Technology standardisation can help
• A bit of integration design up front can save a lot of politics in the future
DESCRIBE-EXECUTE-ANALYSE

Describe

Analyze

Execute
DOG FOOD CONTEXT

Music Streaming Platform

External white label
Internal team

ntcoding
MORE HEURISTICS

• Align with consistency needs
• Minimise overall dependencies
• Who loves me the most?
• Respect operational capabilities
• Minimise social complexity
MORE PATTERNS AND HEURISTICS

- ntcoding.co.uk/org-design
- medium.com/nick-tune-tech-strategy-blog/
IMPLEMENTING BOUNDED CONTEXTS
AVOID SHARING BETWEEN CONTEXTS
Each bounded context should be implemented according to its value and domain complexity.
IMPLEMENTATION PATTERNS

- CRUD
- Domain Model
- Transaction Script
- CQRS & Event Sourcing
CQRS & EVENT SOURCING

- POST: Command Model
- GET: Query Model

Event persistence
Projection persistence
INTEGRATION TIP

COMMUNICATE WITH EVENTS

Communicate between bounded contexts using events to bind strategic model and domain (another reason to practice event storming).
Events are a notification to many subscribers that something has happened. Commands are an imposition to one recipient to perform some action.
Typically we prefer asynchronous integration but it’s not a hard and fast rule.
WHO OWNS THE UI?

- Page owned by 1 BC
- Micro-frontend / composite UI
- BFF / separate frontend team
LEGACY PATTERNS

• Anti-corruption Layer
• Bubble Context
• Autonomous Bubble
• Ice-pick
ALIGNMENT & RESILIENCE: PEOPLE PATTERNS
Perfect boundaries & 100% autonomous teams do not exist and you wouldn’t want them anyway.
RESILIENCE PATTERNS

• Use inner-sourcing to ‘take the people to the work’
• Cross-team pairing
• Rotate people regularly to build empathy and spread domain knowledge
THE NOMAD PATTERN

New people pair with many teams across the org before joining a team.
BECOMING A SOCIOTECHNICAL THINKER
Software developers literally make million dollar decisions without even realising it.
Our challenge when designing software systems is continuously balancing economical, social, and technical factors.
THE 5 PRIMARY SOCIOTECHNICAL DESIGN HEURISTICS

1. 💰 Align with Business Value
2. ☹️ Align with Business Domain
3. ❤️ Optimise for Social Needs
4. 🖥️ Respect Technical Constraints
5. 😍 Optimise for UX & Brand Perception
LET’S KEEP IN TOUCH...

ntcoding.co.uk/workshops
ntcoding.co.uk/blog
ntcoding.co.uk/speaking

@ntcoding
/in/ntcoding
# BONUS CONTENT

EVOLUTIONARY DESIGN
An up-front imposition of a large-scale structure is likely to be costly. As development proceeds, you will almost certainly find a more suitable structure.

— Eric Evans (@ericevans0)
SOCIOTECHNICAL EVOLUTION

Teams (The System of Work)

Build and evolve

Customers (The Market)

Indicate raised expectations to

Software Architecture (The System of Software)

Provides value to

@ntcoding
DISTILLATION PATTERN
SEGREGATING THE CORE

Extract part of an existing bounded context into a separate context based on identifying the part as a high value core domain.
SEGREGATING THE CORE

95% of revenue is generated by customers running ad campaigns on Facebook.

Core domain
SLICE AND SCATTER
SLICE AND SCATTER CHALLENGES

• You may increase dependencies
• You have to break up a happy team
• Each team will now be bigger and have more code to maintain
• Possibly different tech in each context
EXTRACT GENERIC CAPABILITY

Notifications
MITOSIS / GROW & SPLIT

Pattern from Dynamic Reteaming by Heidi Helfand
SPLIT BY REVENUE / ENGAGEMENT

InStreet

InStreet

Ads
Community

💰💍
SPLIT BY...

- Team skills
- Backlog cohesion
- Domain cohesion
- Geographical location
SPREAD AND SLICE
SLICE AND SCALE
EVOLUTIONARY TRIGGERS

• Market demand changes
• Our knowledge of the domain improves
• The organisation evolves
• Technical constraints come and go