Designing a Serverless Application with Domain Driven Design

Susanne Kaiser
Independent Tech Consultant
@suksr
Costs of Poor Software Quality in the US in 2018 (by CISQ report)

$2,840,000,000,000


@suksr
Some Indicators for Poor Software Quality (extracted from CISQ report)

- Increasing defect trend
- Low test coverage
- Large inheritance depth
- High amount of effort to understand piece of code
- Lack of domain knowledge
- High degree of class coupling
- Badly engineered software
- Communication and coordination breakdowns in (large) teams

Domain Driven Design (DDD)

Business Domain → Needs → Strategy → Software Design
Domain Driven Design (DDD) – Terminology

- Strategic Design
- Tactical Design
- Core Subdomain
- Supporting Subdomain
- Generic Subdomain
- Problem Space
- Solution Space
- Bounded Context
- Ubiquitous Language
- Context Maps
- Anti-Corruption Layer
- Shared Kernel
- Open Host Service
- Separate Ways
- Partnership
- Customer-Supplier
- Conformist
- Domain Model
- Entity
- Value Object
- Aggregate
- Repository
- Factory
- Application Service
- Domain Service
- Domain Event
DDD & Wardley Maps
Wardley Maps by Simon Wardley

Visualisation of a value chain’s evolution

Evolution

Value Chain

Visible

Uncharted

Invisible

Position

Movement

Genesis

Custom-Built

Product (+rental)

Commodity (+utility)

Industrialised

@suksr
Who are your users?
Who are your users?

What are your users’ needs?
Wardley Maps – VALUE CHAIN

Who are your users?

What are your users’ needs?

What are the components/activities to fulfill your users’ needs incl. dependencies?
Wardley Maps – LANDSCAPE

Components along evolution axes

Evolution

Value Chain

Position

Movement

Genesis

Custom-Built

Product (+rental)

Commodity (+utility)
Everything evolves

Supply Competition

Demand Competition

Evolution

Past

Current

Future

Visible

Uncharted

Industrialised

Value Chain

Movement

 Genesis  Custom-Built  Product (+rental)  Commodity (+utility)
Wardley Maps – PATTERNS

**Value Chain**
- **Visible**
  - Uncharted
    - Undefined market
      - Uncertain
      - Unpredictable
      - Rare
      - Poorly understood
  - Forming market
    - Learning on use
      - Increasing understanding
      - Slowly increasing consumption
      - Rapid increases in learning
  - Growing market
    - Learning on operation
      - Increasing education
      - Rapidly increasing consumption
      - Rapid increase in use
  - Mature market
    - Known / accepted
      - Stable
      - Widespread and stabilising
      - Commonly understood
        - (in term of use)
    - Industrialised

**Invisible**
- Past
- Movement
- Genesis
- Custom-Built
- Product (+rental)
- Commodity (+utility)

**Evolution**
- Past
- Current
- Future

**Characteristics change**
- Uncharted
- Forming market
- Growing market
- Mature market

**Industrialised**
- Known / accepted
- Stable
- Widespread and stabilising
- Commonly understood
  - (in term of use)
Wardley Maps – PRINCIPLES

Use appropriate methods per evolution stage

Build in-house
Use/buy off-the-shelf product
Outsource to utility suppliers

Evolution

- Genesis
- Custom-Built
- Product (+rental)
- Commodity (+utility)

Value Chain

Visible

Invisible
Wardley Maps – PRINCIPLES

Build in-house
Use/buy off-the-shelf product
Outsource to utility suppliers

Use appropriate methods per evolution stage
Wardley Maps – PRINCIPLES

Use appropriate methods per evolution stage

Options:
- Build in-house
- Use/buy off-the-shelf product
- Outsource to utility suppliers

Evolution:
- Genesis
- Custom-Built
- Product (+rental)
- Commodity (+utility)
Know your users & focus on user needs

Wardley Maps – PRINCIPLES

- Build in-house / Agile
- Use/buy off-the-shelf product / Lean
- Outsource to utility suppliers / Six Sigma

Evolution

Value Chain

Visible

Invisible

Genesis

Custom-Built

Product (+rental)

Commodity (+utility)

@suksr
Domain Driven Design

Understanding the problem domain first before solving it

Problem Domain

Collaboration

Domain Experts

Development Teams

Domain Knowledge

Ubiquitous Language

@suksr
DDD & Wardley Maps

Problem Domain

Value Chain

Visible

Strategic Design

Invisible

Tactical Design

Evolution

Genesis

Custom-Built

Product (+rental)

Commodity (+utility)

@suksr

DDD Patterns & Practices

Strategic Design

Tactical Design
DDD & Wardley Maps

Problem Domain

Discovering Subdomains

Problem Space

Analysing the business domain

Strategic Design

Tactical Design

Value Chain

Visible

Strategic Design

Analyse & Discover

Invisible

Tactical Design

Evolution

Genesis

Custom-Built

Product (+rental)

Commodity (+utility)

@suksr
DD & Wardley Maps

Problem Domain

Value Chain

Visible

Strategic Design

Solution Space

Decompose & Map

Problem Space

Analisng the business domain

Discovering Subdomains

Evolution

Problem Space

High-Level Design Decisions

Decomposing into modular components (Bounded Contexts (BC))

Mapping interaction patterns between BC (Context Maps)

Solution Space

Strategic Design

Tactical Design

Commodity (+utility)

Product (+rental)

Custom-Built

Genesis

@suksr
DDD & Wardley Maps

Problem Domain

Strategic Design

- Analysing the business domain
- Discovering Subdomains
- Decomposing into modular components (Bounded Contexts (BC))
- Mapping interaction patterns between BC (Context Maps)

Tactical Design

- Architecting a solution fitting the problem domain as closely as possible
- Provides building blocks to implement domain model

Value Chain

Visible

- Strategic Design
- Decompose & Discover Map
- Low-Level Design Decisions

Invisible

- Genesis
- Custom-Built
- Product (+rental)
- Commodity (+utility)

Problem Space

Solution Space

Evolution

@suksr
Distilling the problem domain & discovering the core subdomain
DDD & Wardley Maps

STRATEGIC DESIGN (PROBLEM SPACE)

Distilling the problem domain & discovering the core subdomain

Competitive advantage
- Complex
- Changes often
- Build in-house

Build in-house
Use/buy off-the-shelf product
Outsource to utility suppliers

@suksr
Strategic Design (Problem Space)

- **Competitive advantage**
  - Complex
  - Changes often
  - Build in-house

- **No competitive advantage**
  - Quite simple
  - Does not change often
  - Prefer to buy/use off-the-shelf

- **Core Subdomain**
- **Supporting Subdomain**
- **Generic Subdomain**

**Build in-house**
**Use/buy off-the-shelf product**
**Outsource to utility suppliers**

Distilling the problem domain & discovering the core subdomain
DDD & Wardley Maps

STRATEGIC DESIGN (SOLUTION SPACE)

Bounded Contexts
- Model integrity boundary
- Linguistic/semantic boundary
- Ownership boundary
- Physical boundary
- Different architectural patterns per context possible

Value Chain
- Visible
- Invisible

Evolution
- Genesis
- Custom-Built
- Product (+rental)
- Commodity (+utility)

Microservice candidates

Core Subdomain
Supporting Subdomain
Generic Subdomain
Problem Domain

Model integrity boundary
Linguistic/semantic boundary
Ownership boundary
Physical boundary
Different architectural patterns per context possible
Know your users & focus on user needs
Discovering subdomains

DDD & Wardley Maps
STRATEGIC DESIGN (PROBLEM SPACE)

Evolution

Genesis

Custom-Built

Product (+rental)

Commodity (+utility)

Visible

Invisible

Strategic Design

Problem Space

Value Chain

Problem & Discover

Submit Session

Core

Speaker

Manage Event

Evaluate Submissions

Build & Publish Schedule

Communicate w/ Speakers

Signup/Signin

Supporting

Generic

@suksr
DDD & Wardley Maps
STRATEGIC DESIGN (SOLUTION SPACE)

Domain Models

Evolution

Speaker
Organiser

Core
Supporting
Generic

Value Chain
Visible

Invisible

Submit Session
Manage Event
Evaluate Submissions
Build & Publish Schedule
Communicate w/ Speakers
Signup/Signin

Submitted Session
Event
Evaluation
Schedule
Message
Account

Open CfP
CFP Settings
Evaluated Session
Scheduled Session
Recipient

Speaker Profile

Genesis
Custom-Built
Product (+rental)
Commodity (+utility)

Problem Space
Analyse & Discover
Map

@suksr
DDD & Wardley Maps
STRATEGIC DESIGN (SOLUTION SPACE)

Bounded Contexts
- Model integrity boundary
- Linguistic/semantic boundary
- Ownership boundary

Submit Session
Manage Event
Evaluate Submissions
Build & Publish Schedule
Communicate w/ Speakers
Signup/Signin

Core
Supporting
Generic

Submission Handling
Event Mngmnt.
Session Evaluation
Schedule Mngmnt.
Messaging
Account Handling

Evolution
@suksr
**DDD & Wardley Maps**

STRATEGIC DESIGN (SOLUTION SPACE)

- **Core**
  - Submit Session
  - Manage Event
  - Evaluate Submissions
  - Build & Publish Schedule
  - Communicate w/ Speakers
  - Signup/Signin

- **Supporting**
  - Event Mngmnt.
  - Session Evaluation
  - Schedule Mngmnt.
  - Messaging
  - Account Handling

- **Generic**
  - Submission Handling
  - Artefact Management
  - Data store
  - SCM
  - CI/CD

**Bounded Contexts**

- Model integrity boundary
- Linguistic/semantic boundary
- Ownership boundary
- Physical boundary

**Evolution**

@suksr
Architectural patterns can differ per Bounded Context, e.g.,

- **Layered Architecture**
- **Hexagonal Architecture**
- **CQRS**
Business logic implementation patterns can differ per Bounded Context, e.g.

- Domain Model
- Active Record
- Transaction Script
DDD & Wardley Maps

TACTICAL DESIGN

Building Blocks of Domain Models

- Value Object
- Entity
- Aggregate
- Repository
- ApplicationService
- Domain Event
- ...

Submission Handling
Submit Session
Manage Event
Evaluate Submissions
Build & Publish Schedule
Communicate w/ Speakers
Signup/Signin

Speaker
Organiser

Evolution

Value Chain

Visible

Invisible

Genesis
Custom-Built
Product (+rental)
Commodity (+utility)

Solution Space
Strategic Design
Tactical Design

Comprise & Decompose & Map
Analyze & Discover
Problem Space
Example Domain Model

**BOUNDED CONTEXT: EVENT MANAGEMENT**

- **Entity**
  - **Event**
    - create: Event
    - publish
    - reschedule
    - rename
    - EventId
      - id: string
    - Name
      - name: string
      - create: Name
    - EventStatus
      - CREATED
      - ACTIVATED
      - DEACTIVATED
  - **Description**
    - desc: string
    - create: Description
  - **Period**
    - start: Date
    - end: Date
    - create: Period

- **Aggregate**
  - **EventModel**
Backend-API w/ Serverless

BOUNDARY CONTEXT: EVENT MANAGEMENT

REST-API with AWS API-Gateway and AWS Lambda

AWS API Gateway

EventController

EventApplicationService

Application

Business Logic

Outer

Inner

Outside

Port

Adapter

POST /events
DELETE /events/{id}
POST /events/{id}/publish

newEvent
delevent
publishEvent

EventController

@suksr
import express from 'express';
import http from 'http';
import path from 'path';
import 'core-js/stable';
import 'regenerator-runtime/runtime';

const app = express();

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.post('/api/events', async (req, res) => {
  // ... process request
  res.status(201).json({ message: 'Event created' });
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
Backend-API w/ Serverless

BOUNDDED CONTEXT: EVENT MANAGEMENT

AWS API Gateway

EventController
(Adapter)

EventApplicationService
(Port)

Application

Event

EventRepository
(Port)

DynamoDBEventRepository
(Adapter)

@suksr
export default class EventApplicationService {
  private readonly eventRepository: EventRepository;
  constructor(eventRepository: EventRepository) {
    this.eventRepository = eventRepository;
  }
  public async publishEvent(id: EventId): Promise<void> {
    const event = await this.eventRepository.eventOfId(id);
    if (!event) {
      throw new Error("Could not publish event with id " + id + ", since event does not exist.");
    }
    event.publish();
    await this.eventRepository.saveEvent(event);
  }
  public async newEvent(command: NewEventCommand): Promise<EventId> {
    // ...
  }
}
Domain Model

BC: EVENT MANAGEMENT

AWS API Gateway

EventController (Adapter)

EventApplicationService (Port)

EventRepository (Port)

DynamoDBEventRepository (Adapter)

Event

- create: Event
- publish
- reschedule
- deactivate
- rename

EventId
- id: string

Name
- name: string

EventStatus
- CREATED
- ACTIVATED
- DEACTIVATED

Description
- create: Description

Period
- create: Period

Entity

Eventid
- id: string

Name
- name: string

EventStatus
- CREATED
- ACTIVATED
- DEACTIVATED

Description
- desc: string

Period
- start: Date
- end: Date

Value Object

Entity

Eventid
- id: string

Name
- name: string

EventStatus
- CREATED
- ACTIVATED
- DEACTIVATED

Description
- desc: string

Period
- start: Date
- end: Date

Aggregate Root

Event

- create: Event
- publish
- reschedule
- deactivate
- rename

EventId
- id: string

Name
- name: string

EventStatus
- CREATED
- ACTIVATED
- DEACTIVATED

Description
- desc: string

Period
- start: Date
- end: Date

Value Object

Entity

Eventid
- id: string

Name
- name: string

EventStatus
- CREATED
- ACTIVATED
- DEACTIVATED

Description
- desc: string

Period
- start: Date
- end: Date
export default class Event {
  readonly id: EventId;
  name: Name;
  description?: Description;
  status: EventStatus;
  period: Period;

  private constructor(id: EventId, name: Name, status: EventStatus, period: Period, description?: Description) {
    this.id = id;
    this.name = name;
    this.description = description;
    this.status = status;
    this.period = period;
  }

  public publish() {
    if (this.status === EventStatus.CLOSED) {
      throw new ValidationError("status", "You cannot publish a closed event");
    }
    if (this.status === EventStatus.PUBLISHED) {
      throw new ValidationError("status", "This event has already been published");
    }
    this.status = EventStatus.PUBLISHED;
  }

  public static create(id: EventId, name: Name, period: Period, status?: EventStatus, description?: Description): Event {
    // ... //
  }
}
export default interface EventRepository {
  saveEvent(event: Event): void;
  eventId(id: EventId): Promise<Event | null>;
  // ... //
}

AWS API Gateway

EventController

DynamoDBEventRepository

Application

EventApplicationService
Data Storage with AWS DynamoDB
export default class DynamoDBEventRepository implements EventRepository {
	nothing

private static TABLE_NAME: string = "events";
private readonly dynamoDbClient: AWS.DynamoDB.DocumentClient;

constructor() {
	this.dynamoDbClient = new AWS.DynamoDB.DocumentClient();
}

public saveEvent(event: Event) {

class

definition

const params: DocumentClient.PutItemInput = {


database adapter

};

return this.dynamoDbClient.put(params).promise();
}

public async eventOfId(id: EventId): Promise<Event|null> {

// ... //
}
DDD & Wardley Maps

TACTICAL DESIGN

- Build in-house
- Use/buy off-the-shelf product
- Outsource to utility suppliers

DDD suits best for the core subdomain

@suksr
Decomposing into Modular Components

Collaboration

Domain Experts

Development Teams

Domain Knowledge

Ubiquitous Language

Gaining Domain Knowledge

Business Domain Needs Strategy

Better Software Design

Aligning Software Design to Business Domain

Discovering the Core Subdomain

Core Subdomain

Do not apply DDD everywhere!
Focus on your core!

DDD helps with ...

@suksr
Some References

https://learnwardleymapping.com/
https://medium.com/wardleymaps
https://miro.com/blog/wardley-maps-whiteboard-canvas/
https://github.com/wardley-maps-community/awesome-wardley-maps
THANK YOU

Susanne Kaiser
Independent Tech Consultant
@suksr
susanne@kaiser-consulting.net