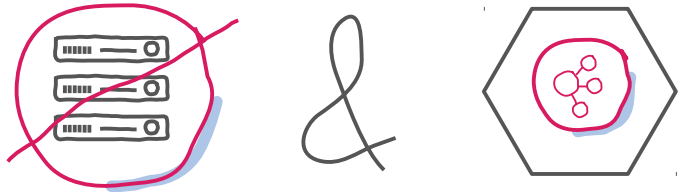


Designing a Serverless Application with Domain Driven Design

Susanne Kaiser
Independent Tech Consultant
@suksr

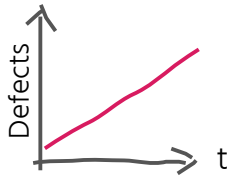
Costs of Poor Software Quality in the US in 2018 (by CISQ report)

$2,840,000,000,000

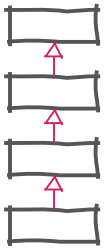
TWOTRILLIONEIGHTHUNDREDFOURTYBILLION USD

Source: https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report

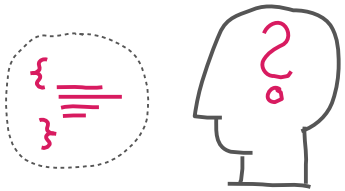
Some Indicators for Poor Software Quality (extracted from CISQ report)



Increasing defect trend

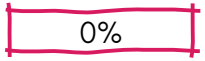


Large inheritance depth

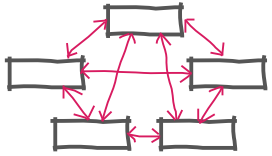


High amount of effort to understand piece of code

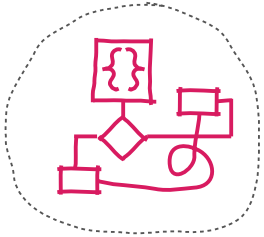
Lack of domain knowledge



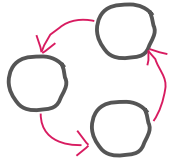
Low test coverage



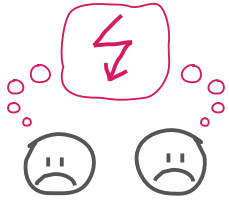
High degree of class coupling



Badly engineered software



Cyclomatic complexity

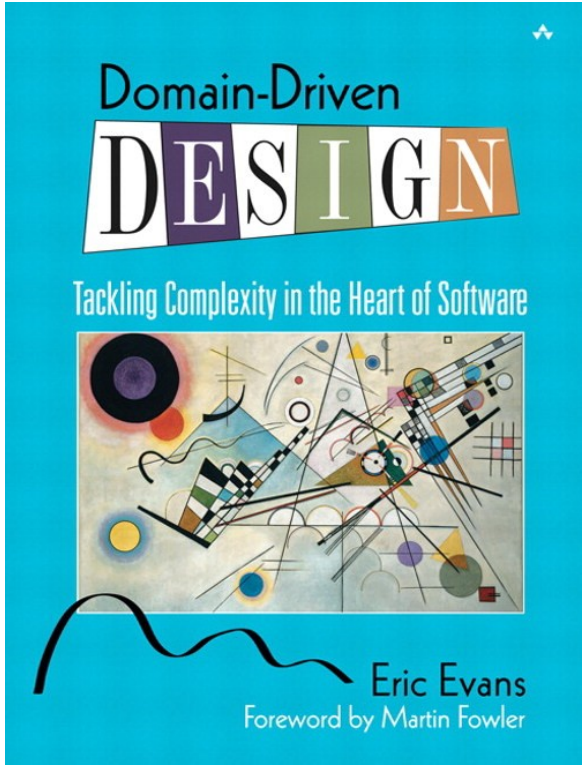


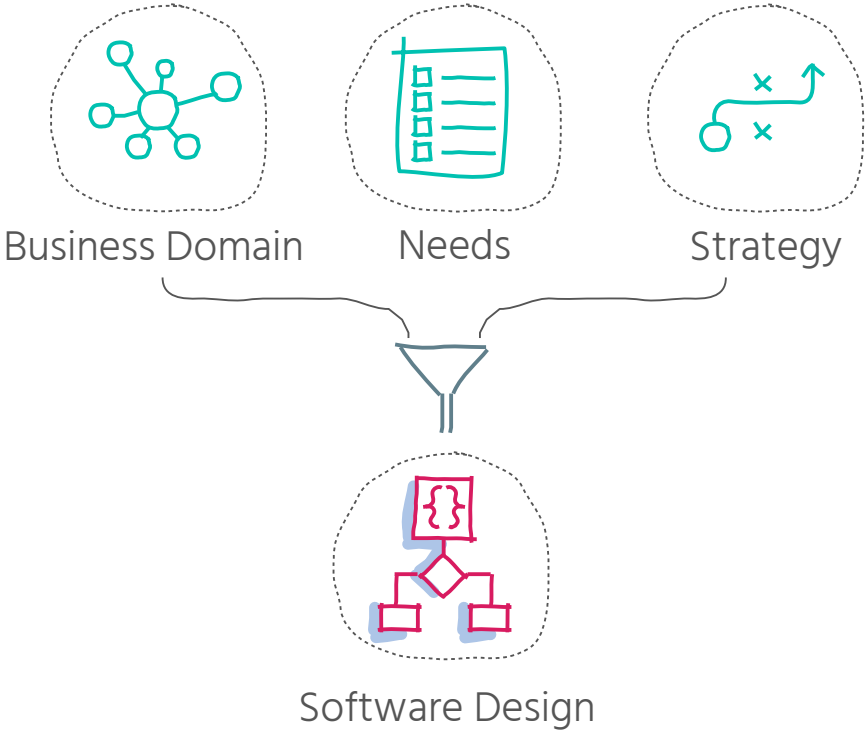
Communication and coordination breakdowns in (large) teams

Based on: https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report

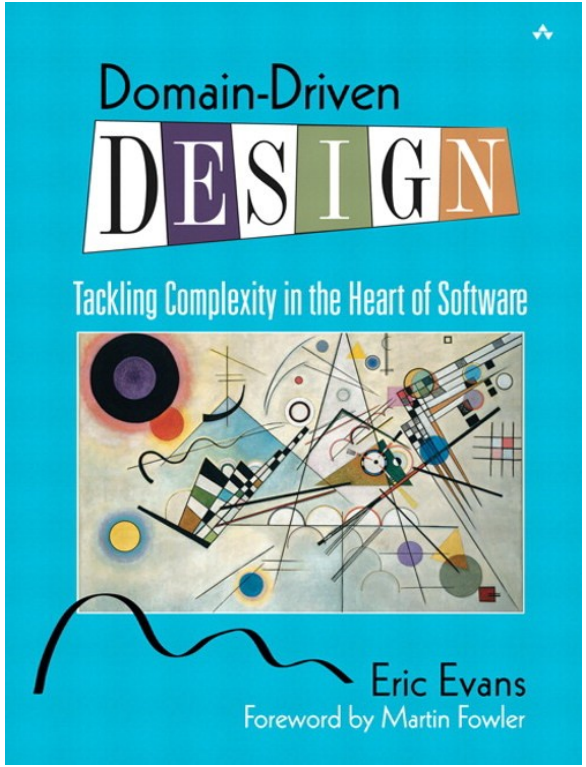
@suksr

Domain Driven Design (DDD)





Domain Driven Design (DDD) – Terminology



Strategic Design
Tactical Design

Problem Space
Solution Space

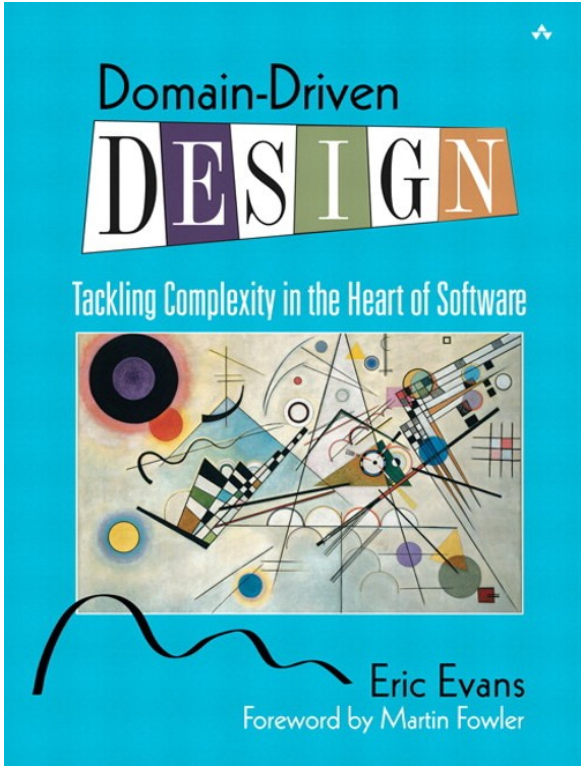
Core Subdomain
Supporting Subdomain
Generic Subdomain

Bounded Context
Ubiquitous Language

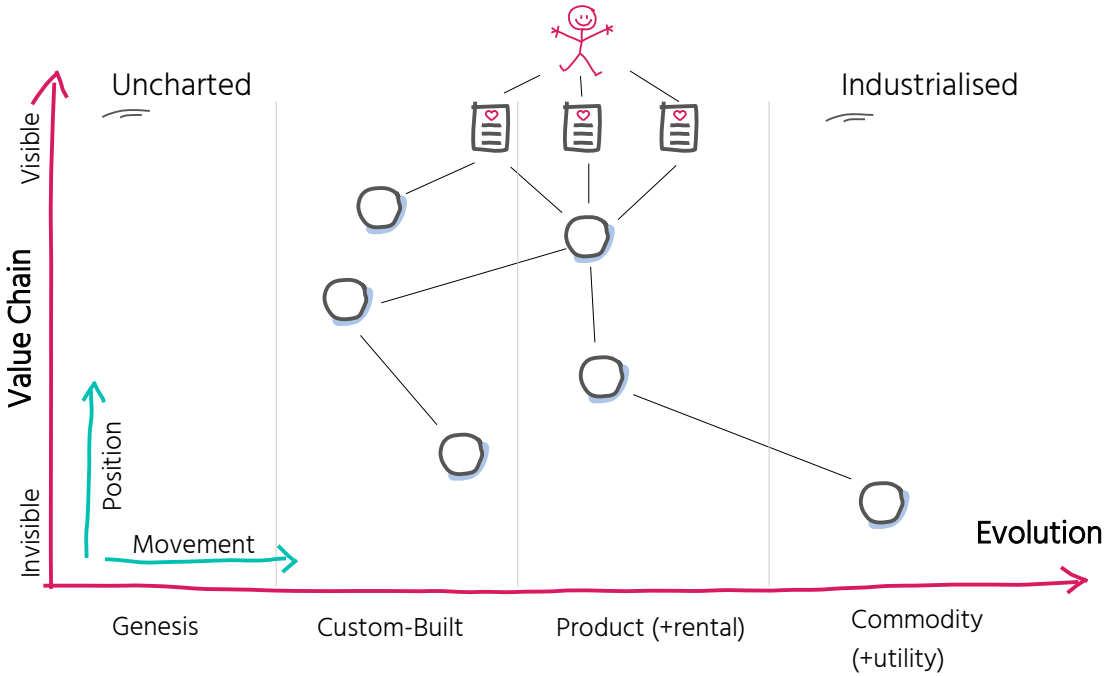
Context Maps
Anti-Corruption Layer
Shared Kernel
Open Host Service
Separate Ways
Partnership
Customer-Supplier
Conformist

Domain Model
Entity
Value Object
Aggregate
Repository
Factory
Application Service
Domain Service
Domain Event

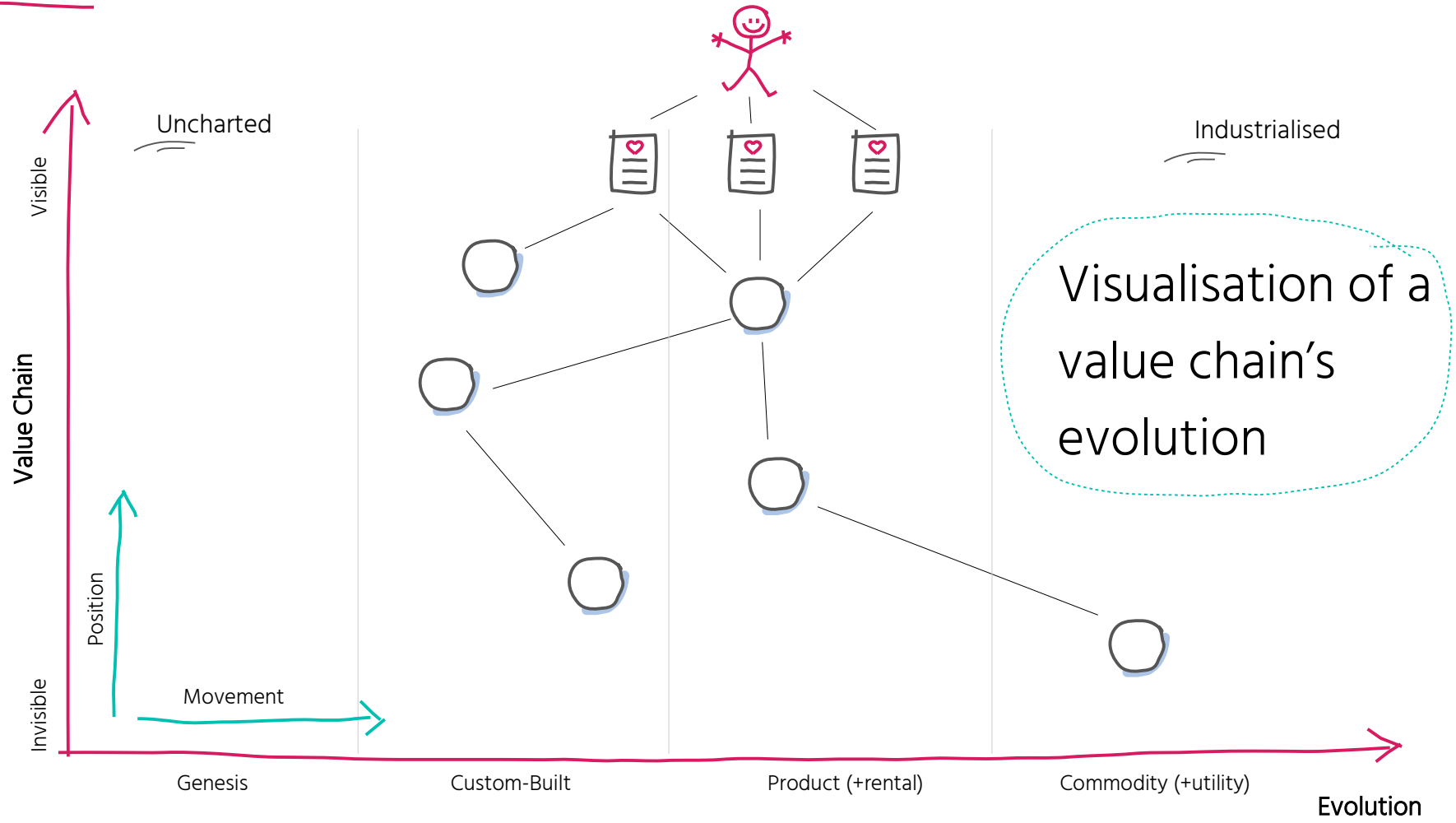
DDD & Wardley Maps



&



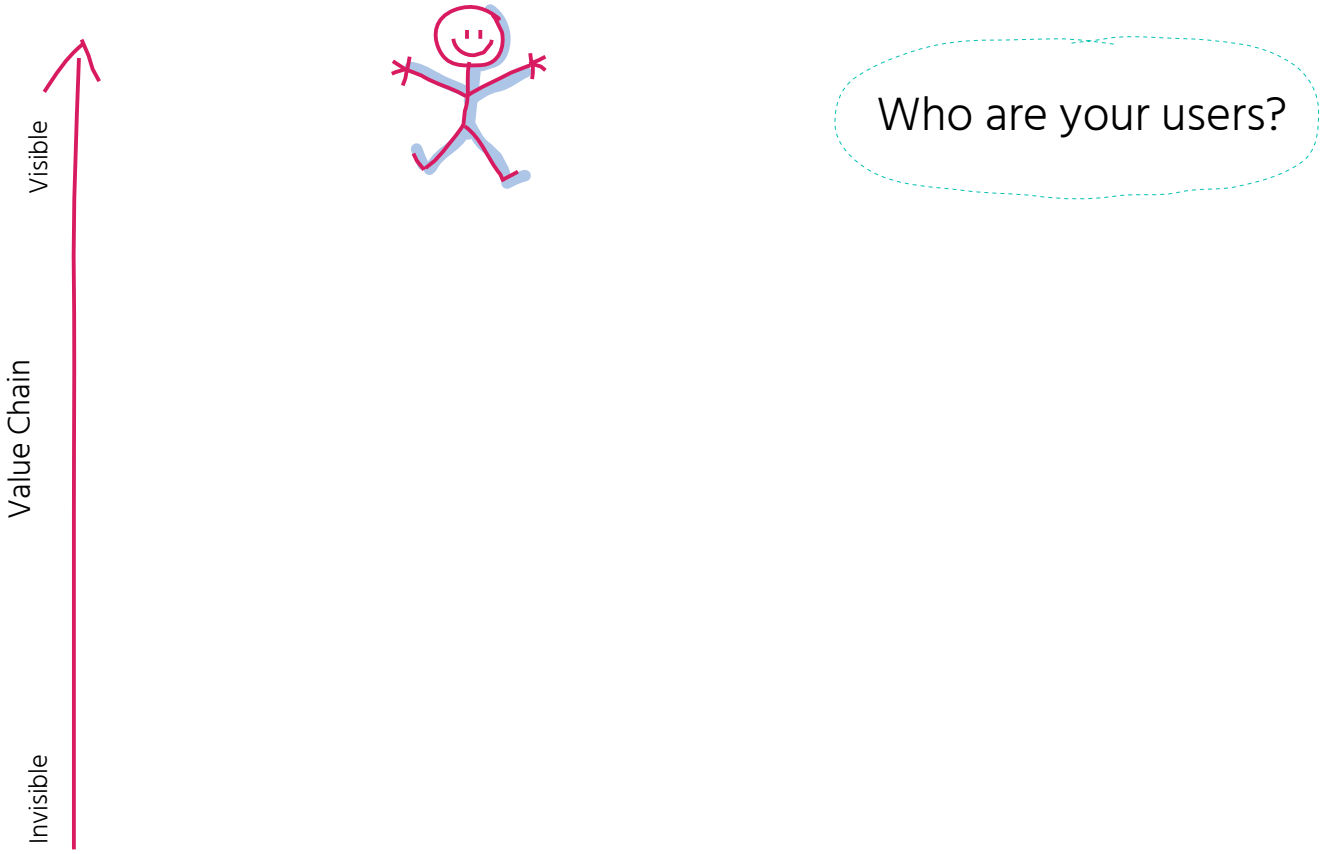
Wardley Maps BY SIMON WARDLEY



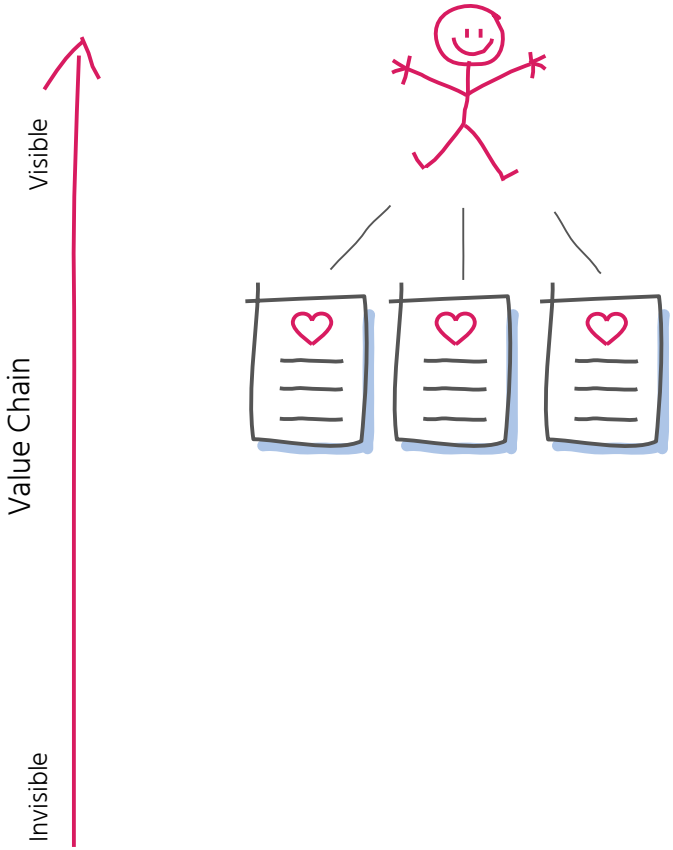
Visualisation of a value chain's evolution

@suksr

Wardley Maps – VALUE CHAIN



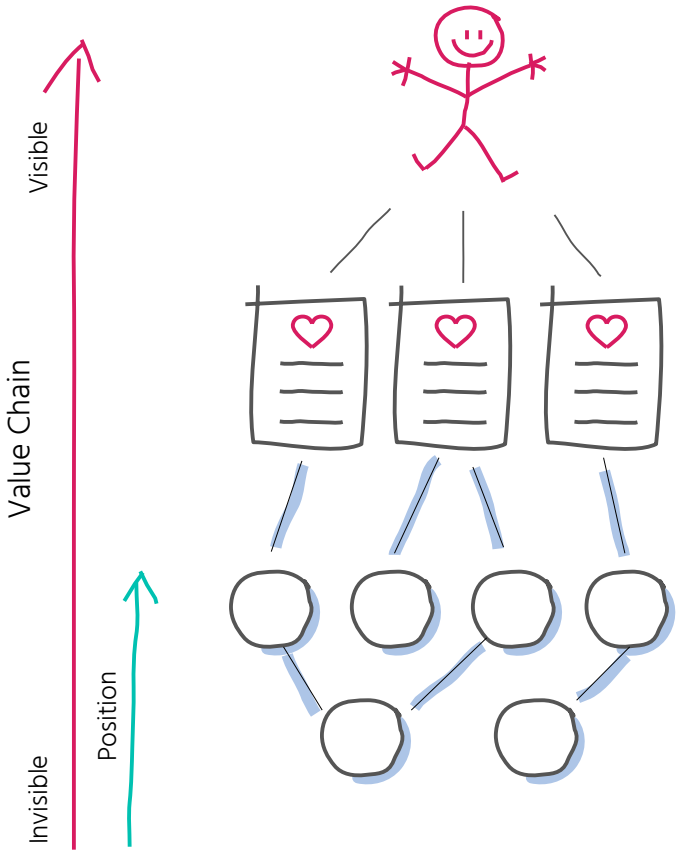
Wardley Maps – VALUE CHAIN



Who are your users?

What are your users' needs?

Wardley Maps – VALUE CHAIN



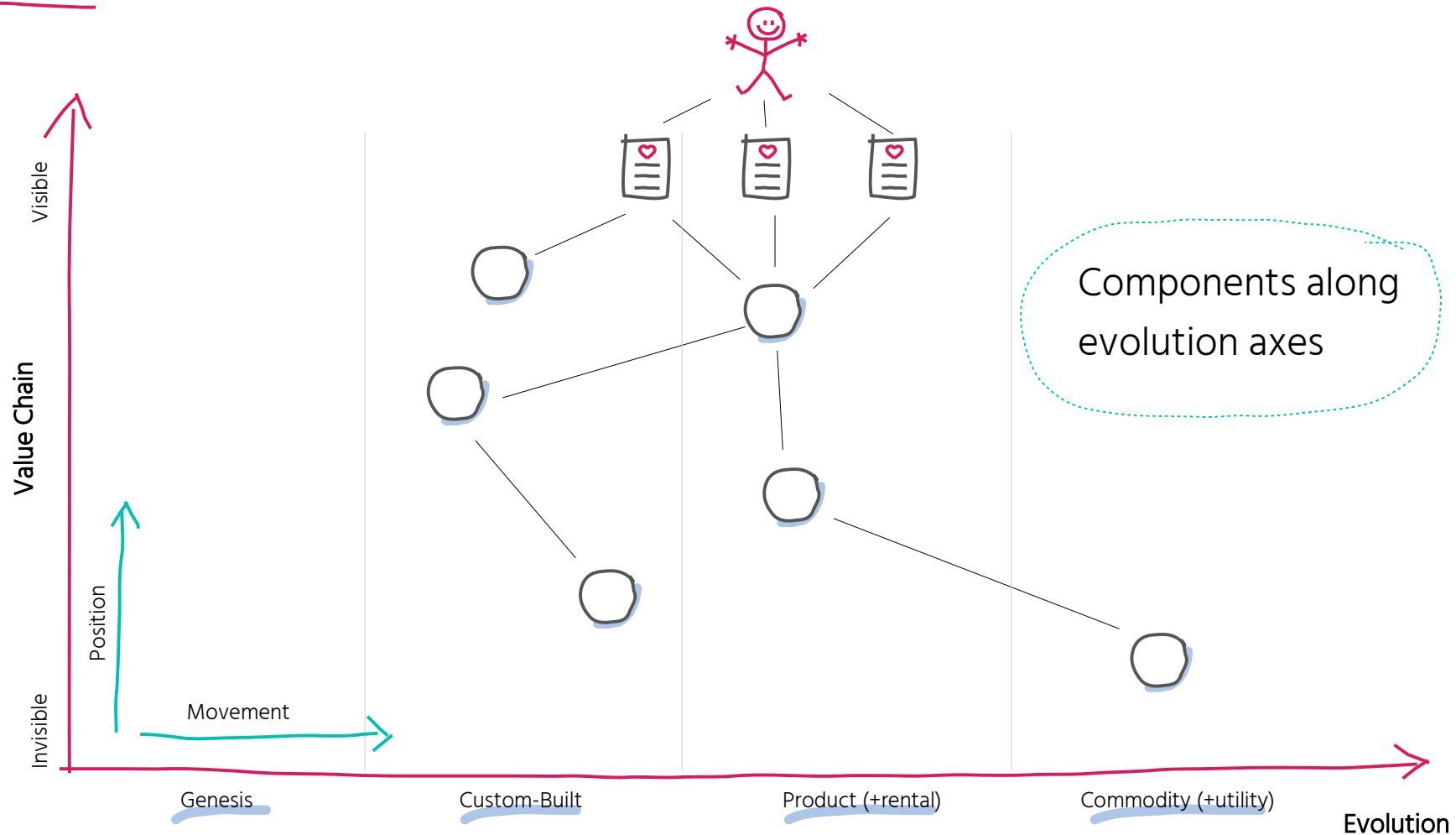
Who are your users?

What are your users' needs?

What are the components/activities to fulfill your users' needs incl. dependencies?

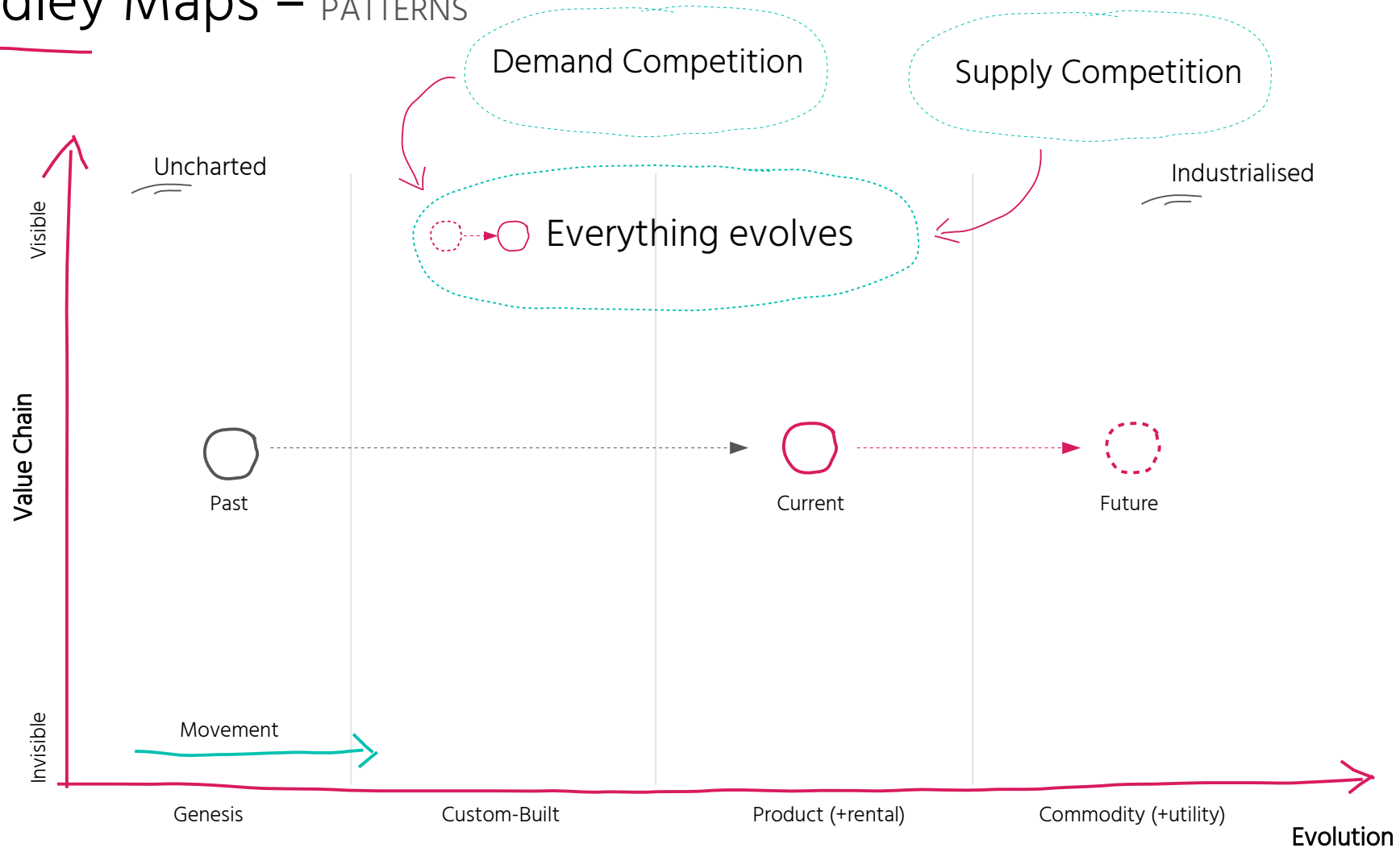
@suksr

Wardley Maps – LANDSCAPE



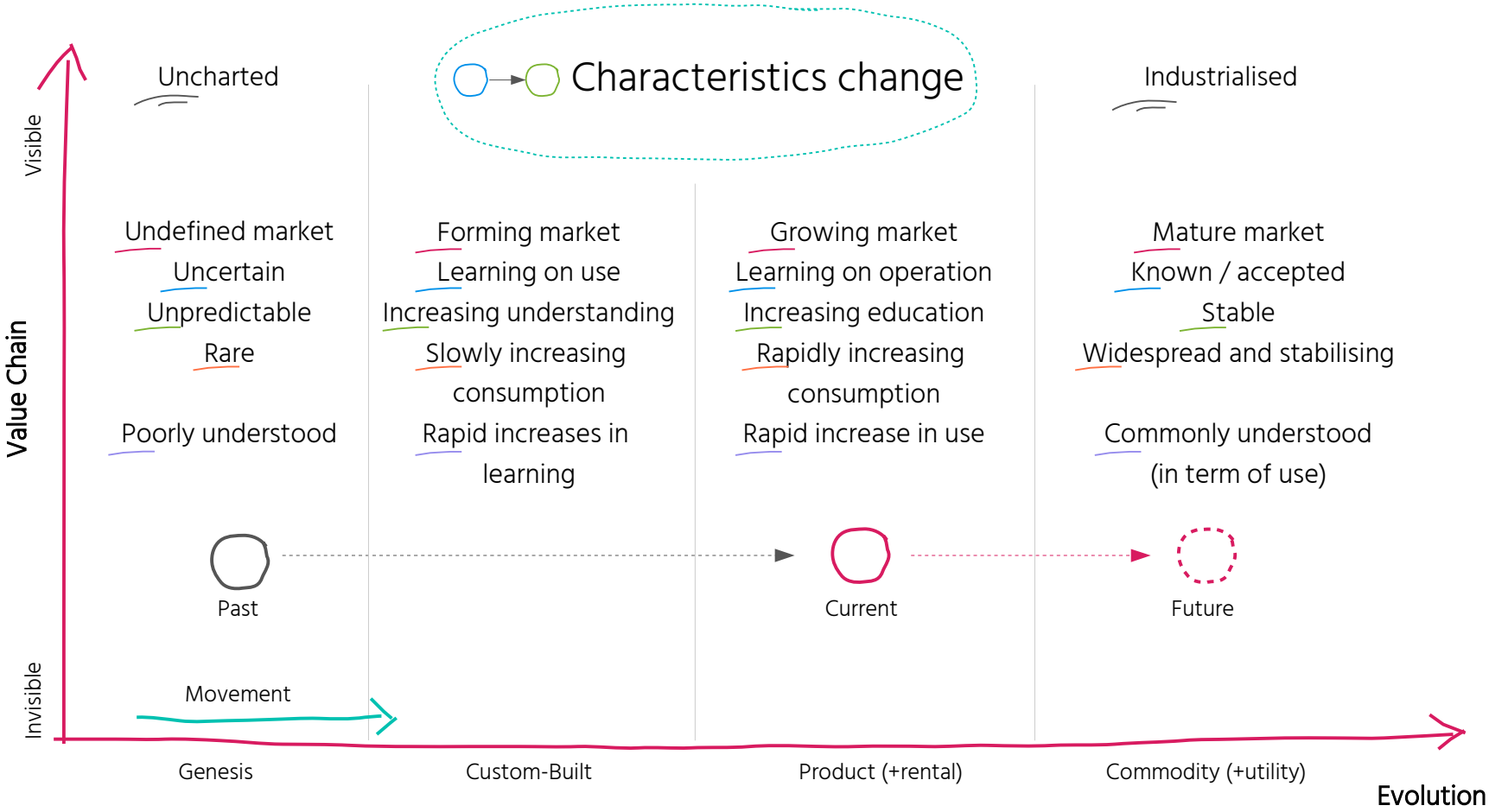
@suksr

Wardley Maps – PATTERNS



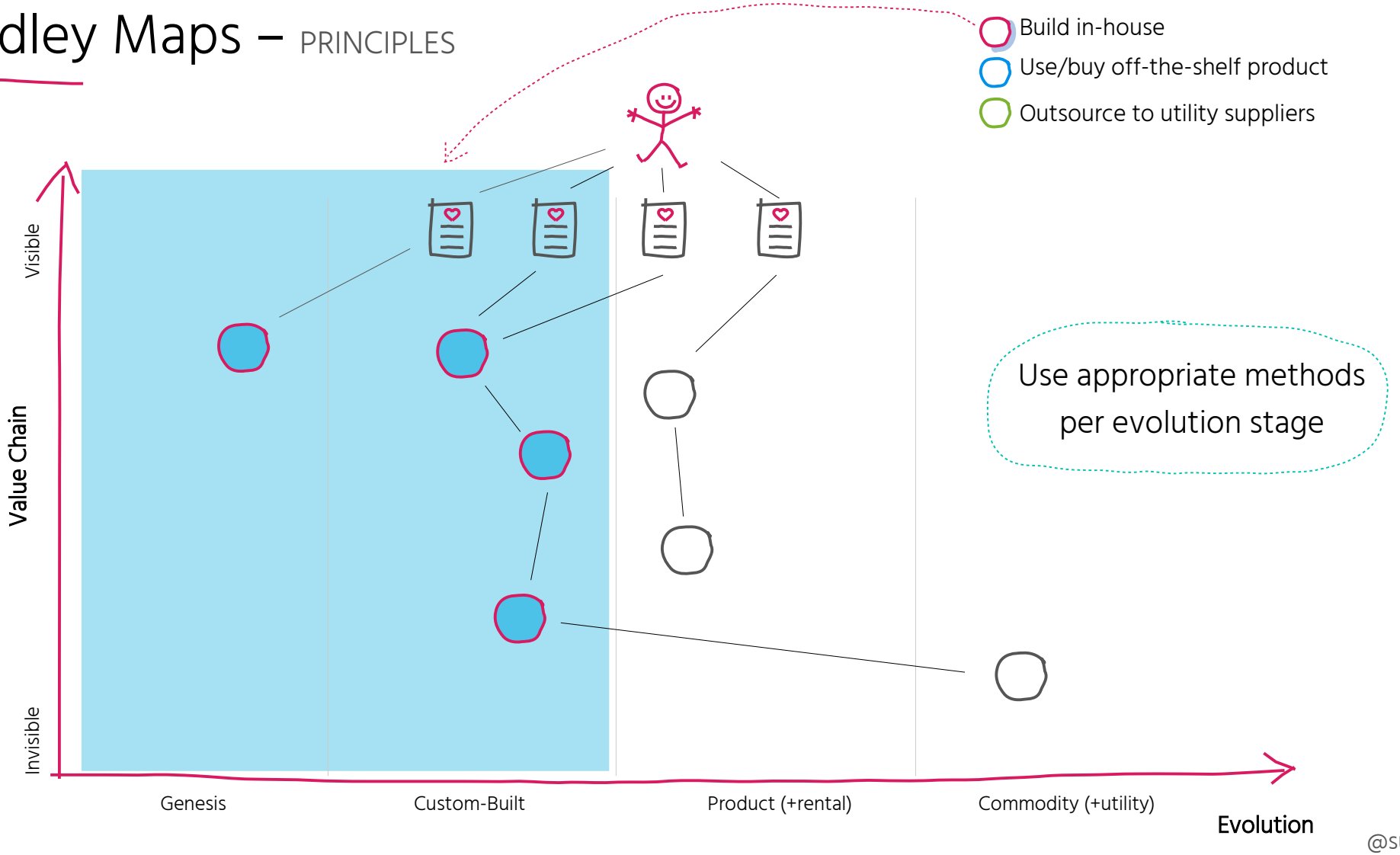
@suksr

Wardley Maps – PATTERNS



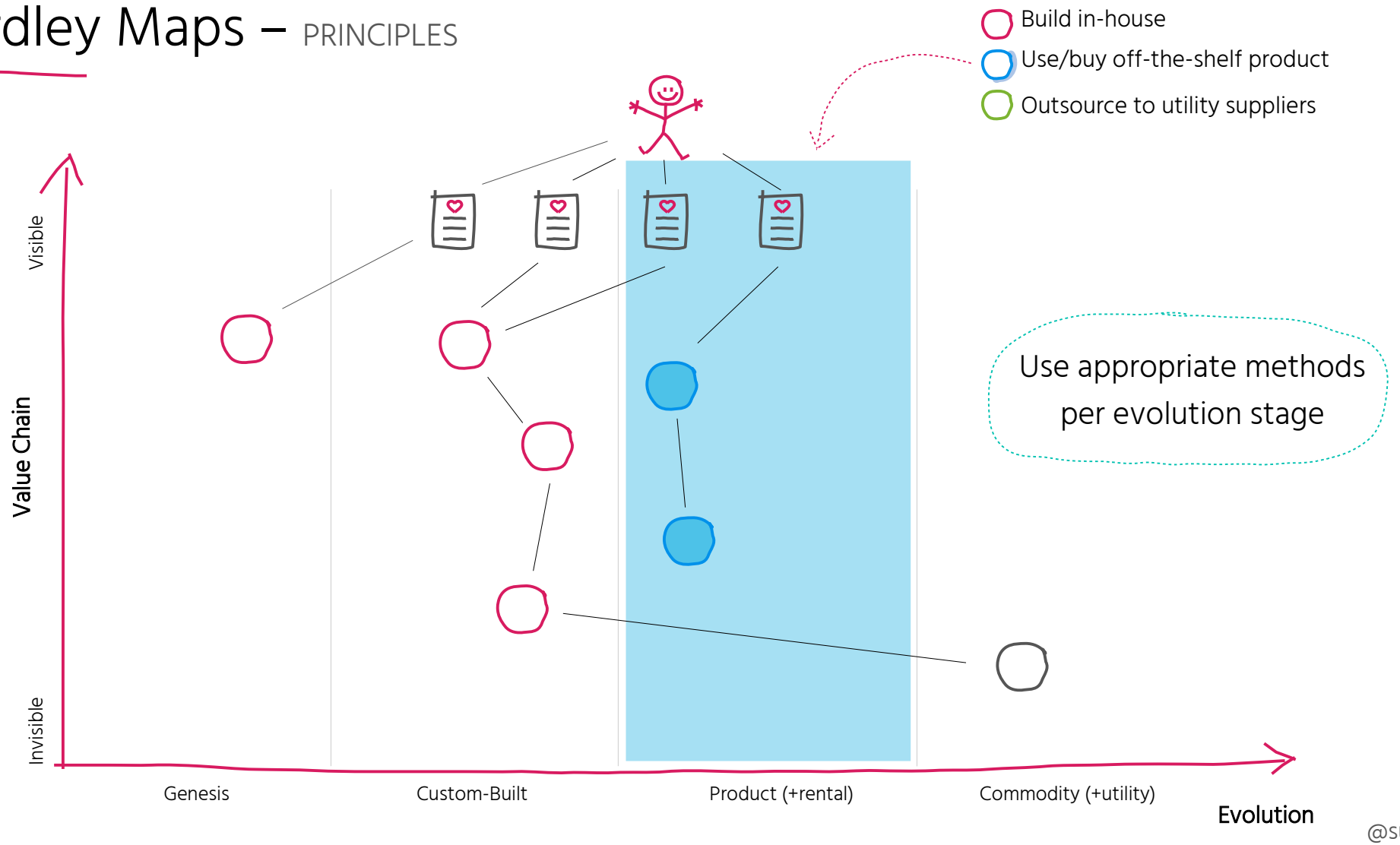
@suksr

Wardley Maps – PRINCIPLES



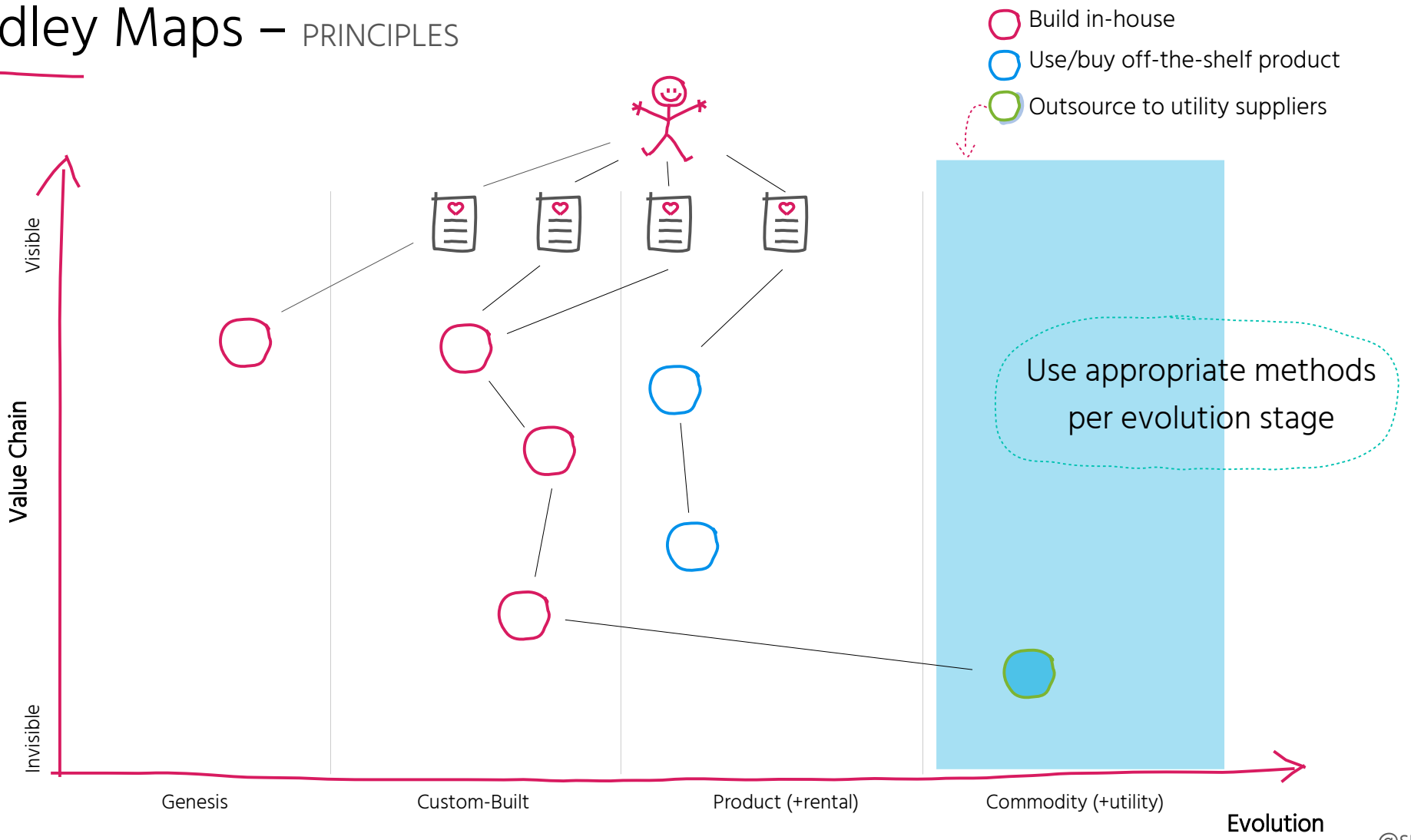
@suksr

Wardley Maps – PRINCIPLES



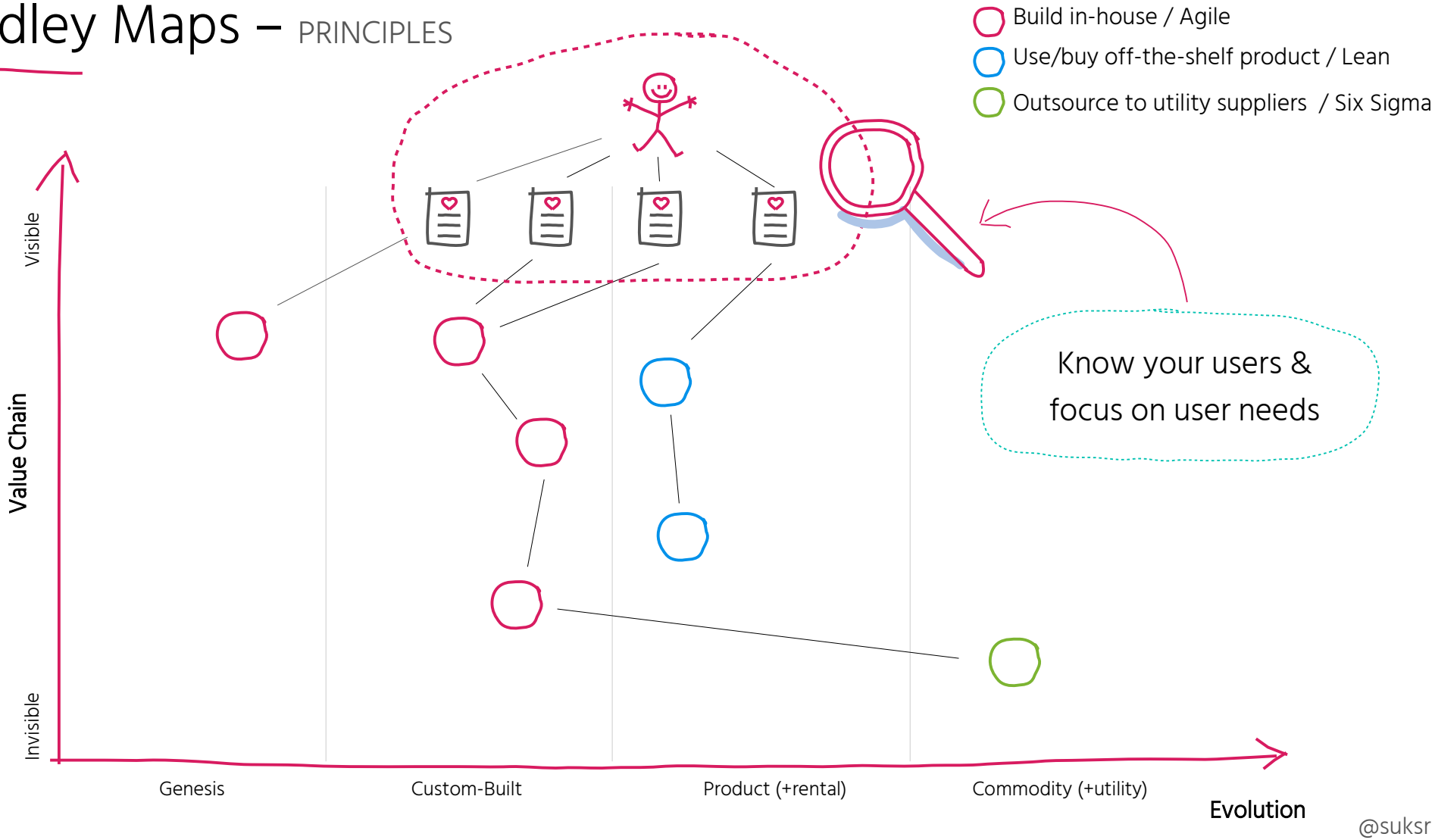
@suksr

Wardley Maps – PRINCIPLES



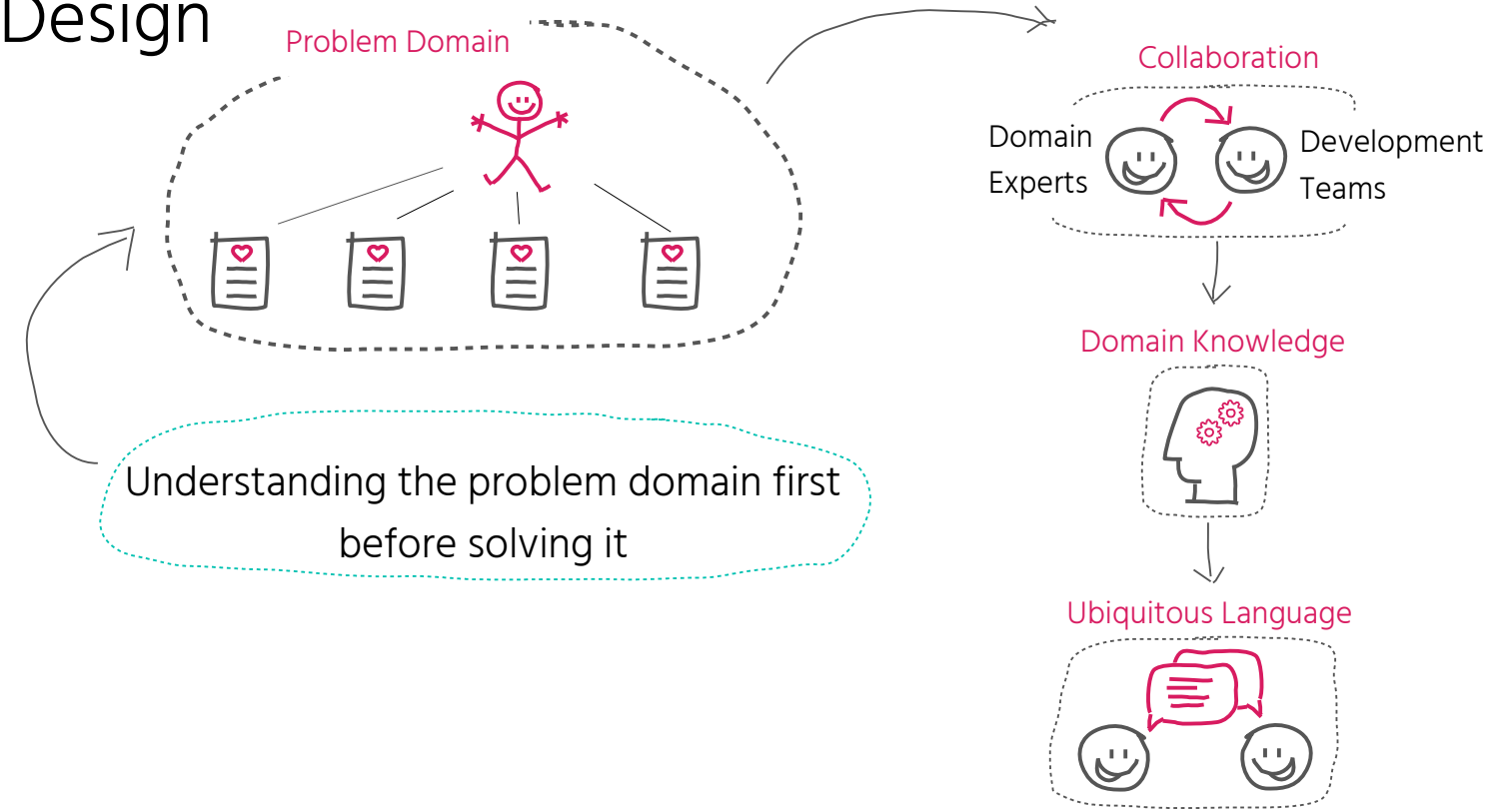
@suksr

Wardley Maps – PRINCIPLES



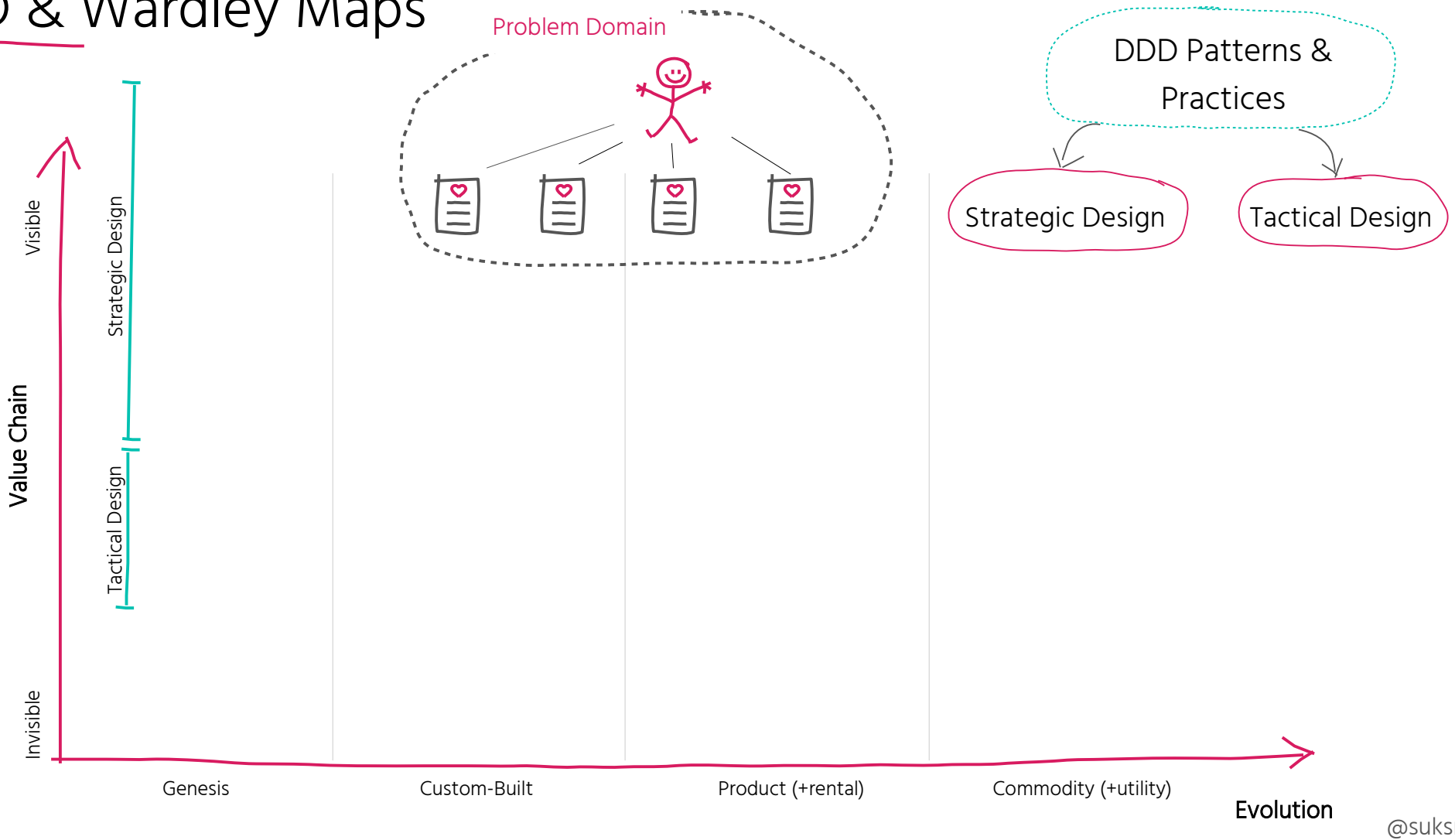
@suksr

Domain Driven Design



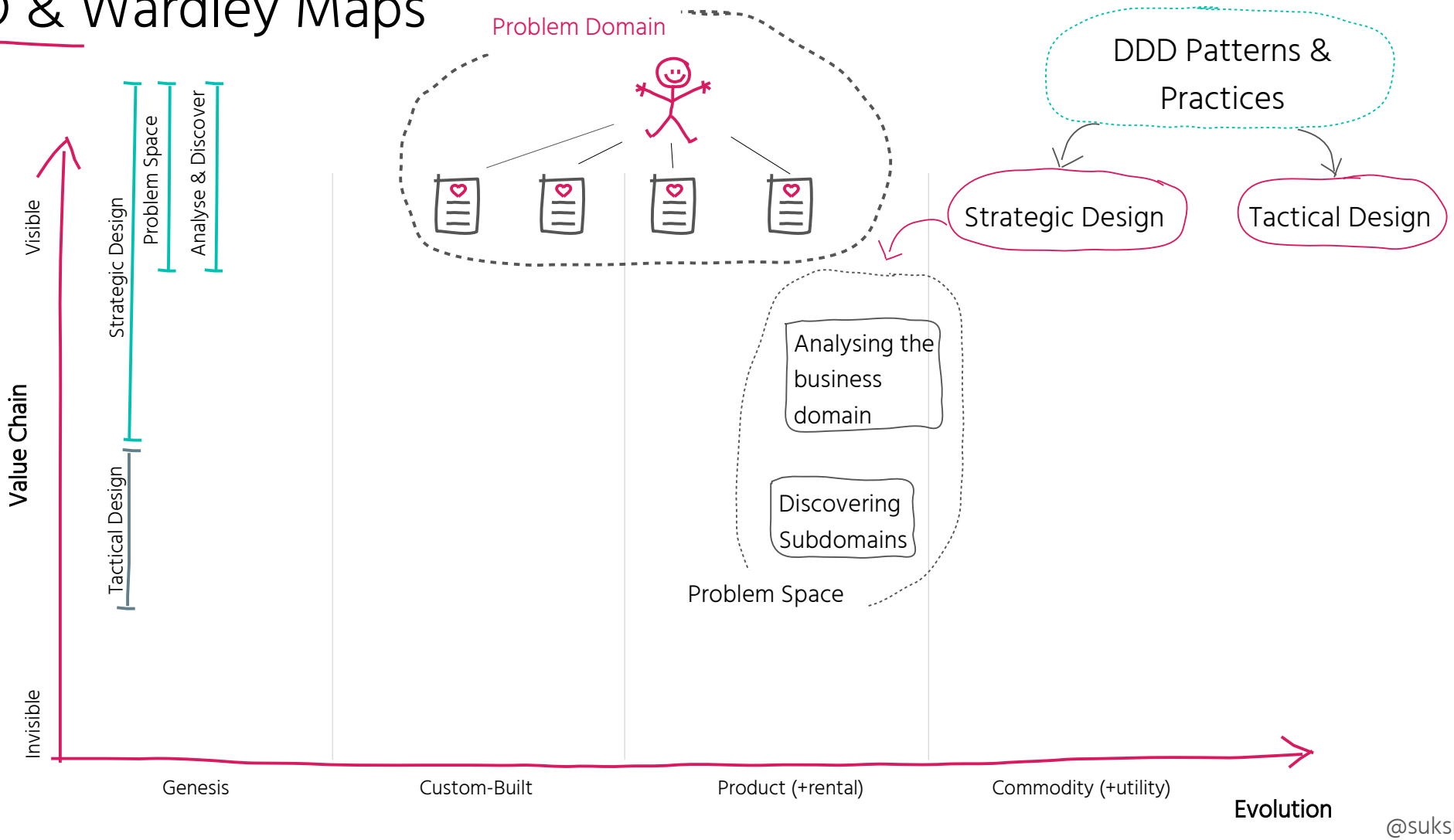
@suksr

DDD & Wardley Maps



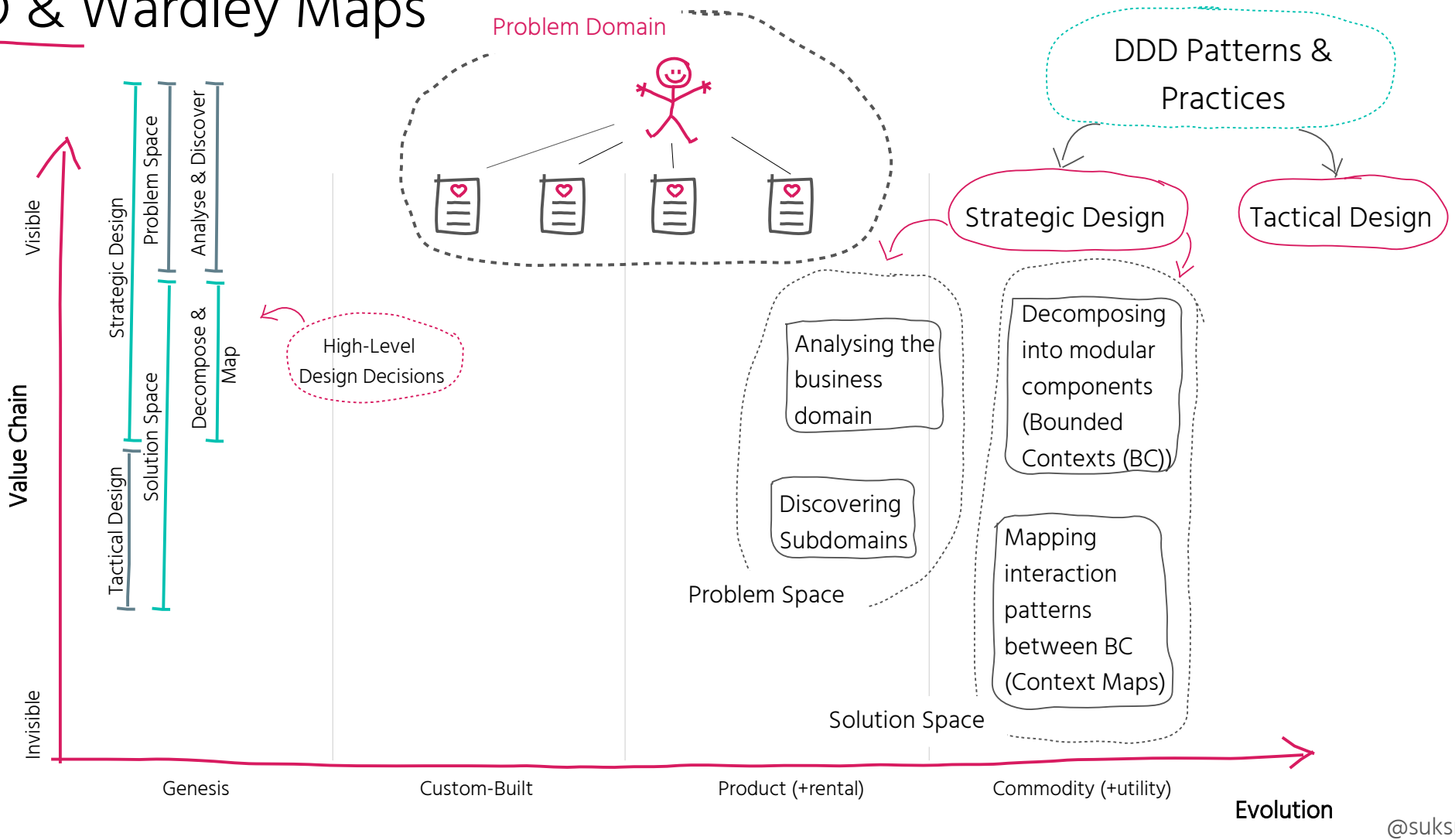
@suksr

DDD & Wardley Maps



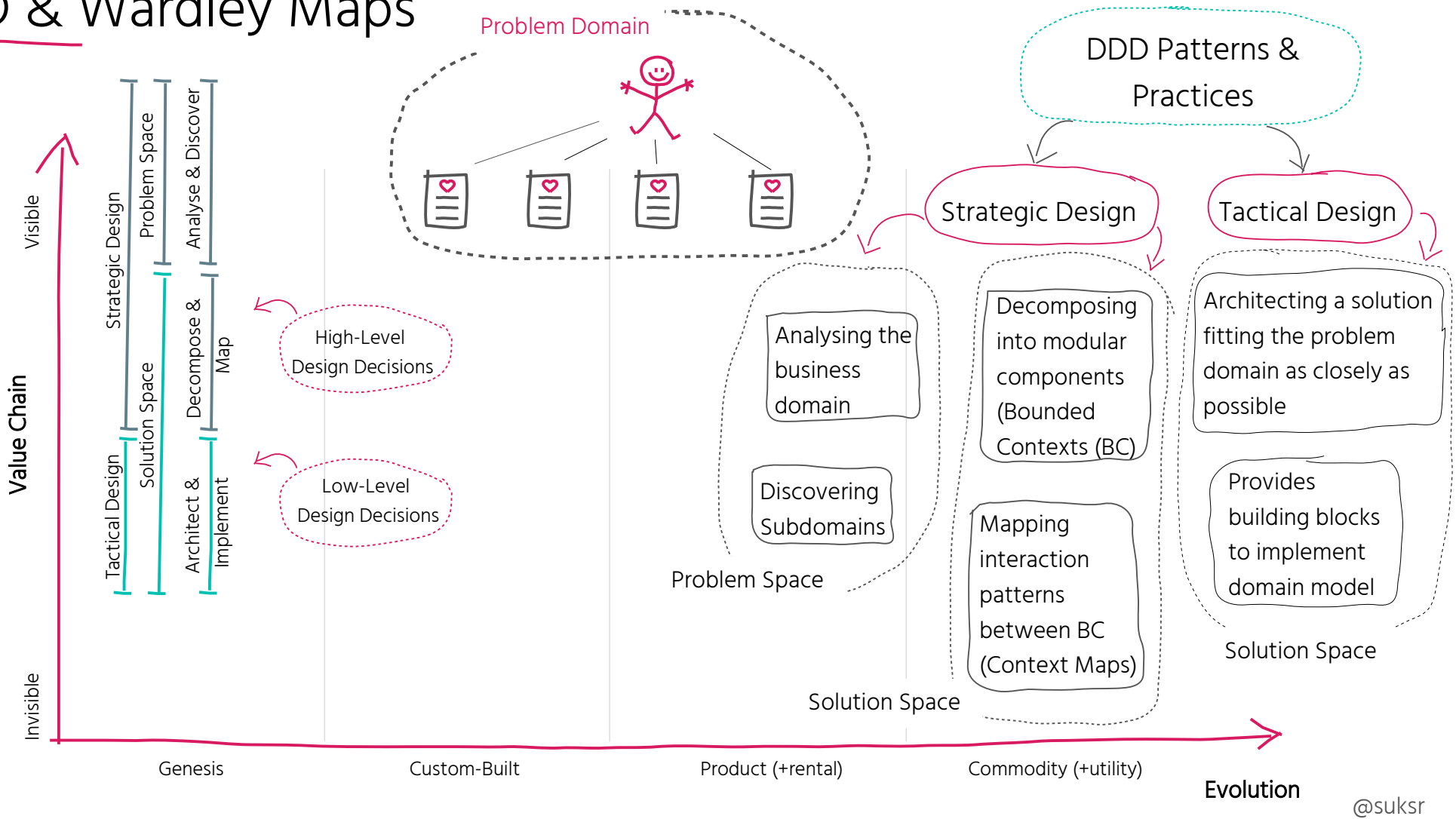
@suksr

DDD & Wardley Maps



@suksr

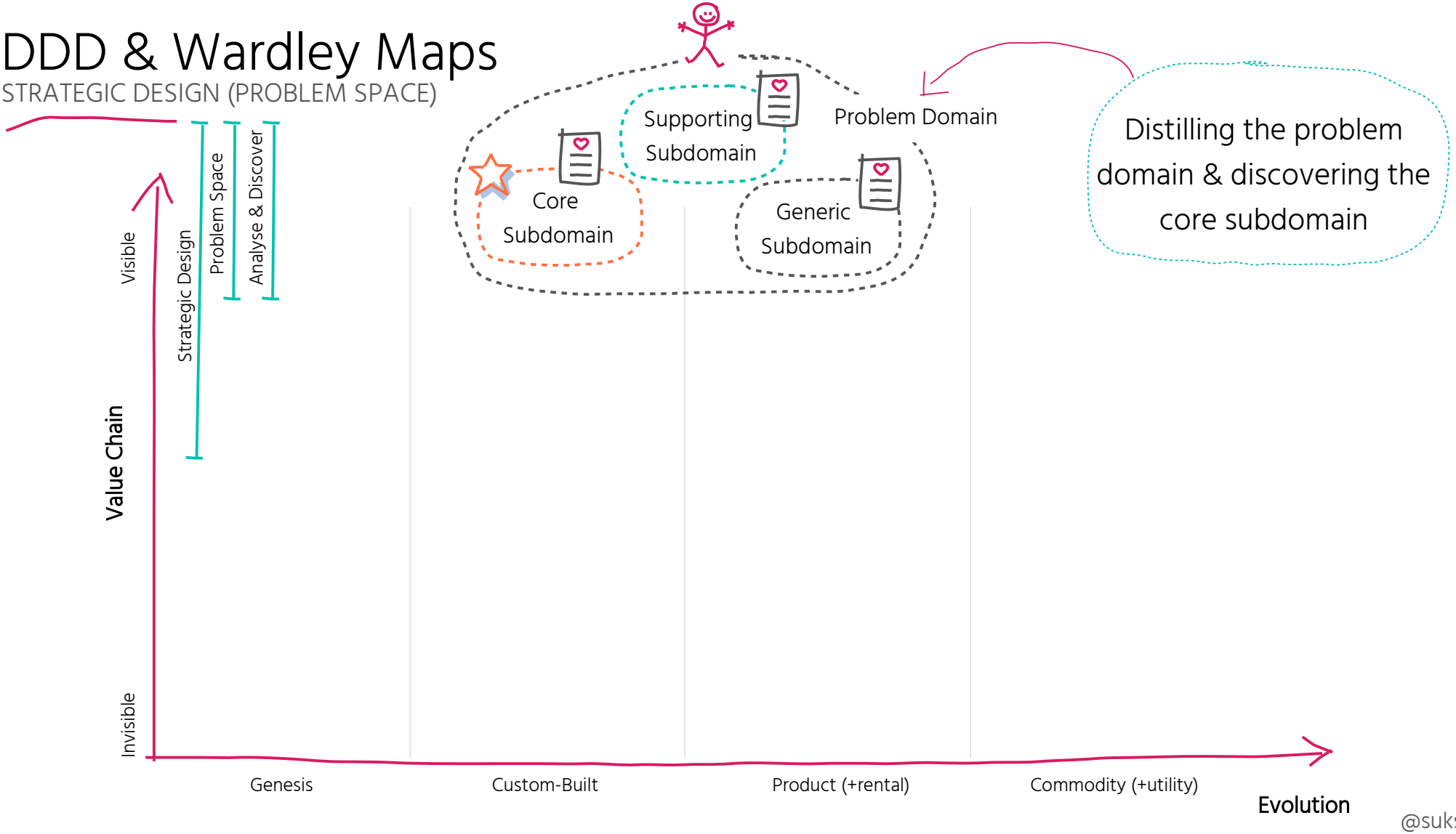
DDD & Wardley Maps



@suksr

DDD & Wardley Maps

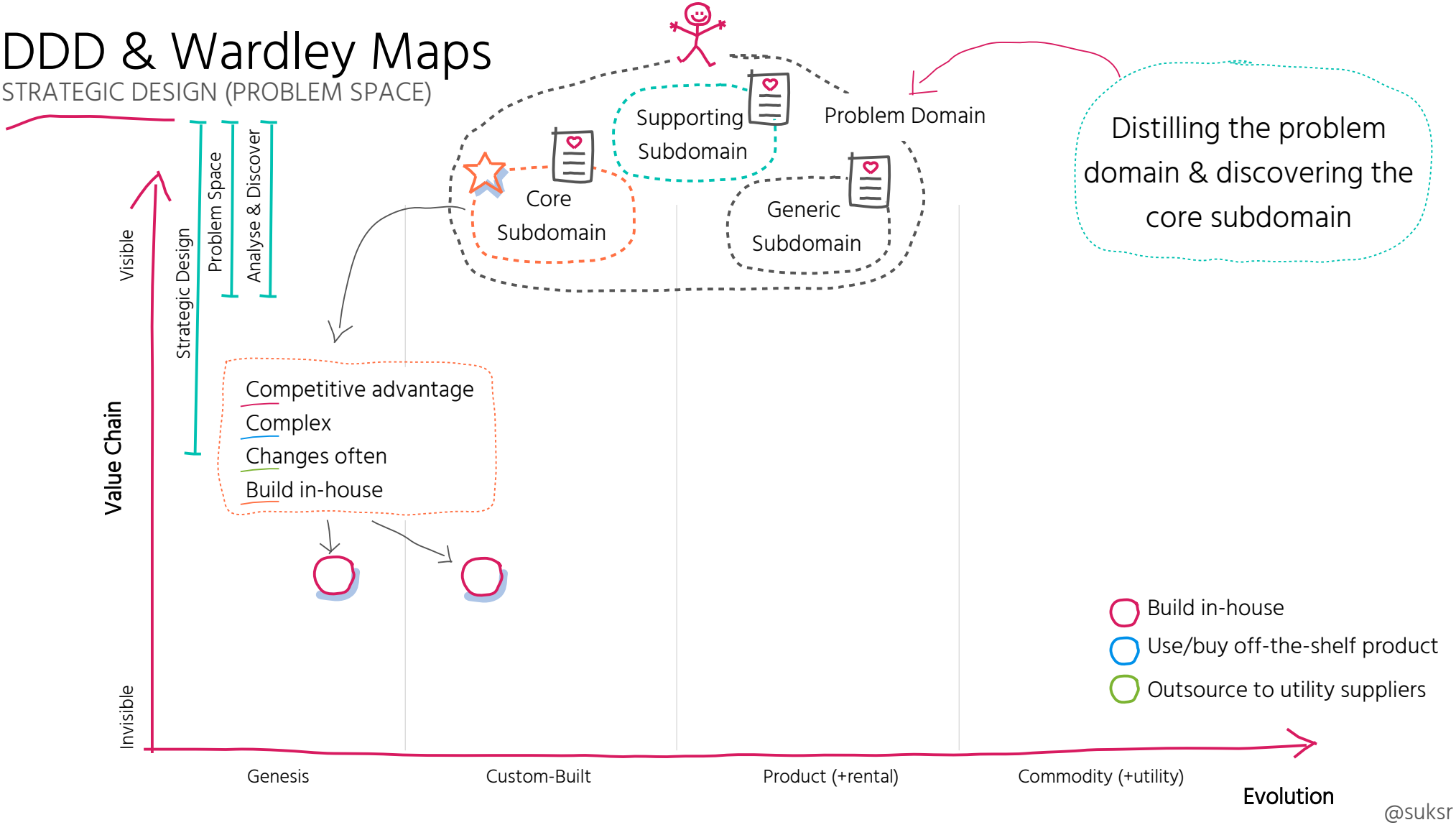
STRATEGIC DESIGN (PROBLEM SPACE)



@suksr

DDD & Wardley Maps

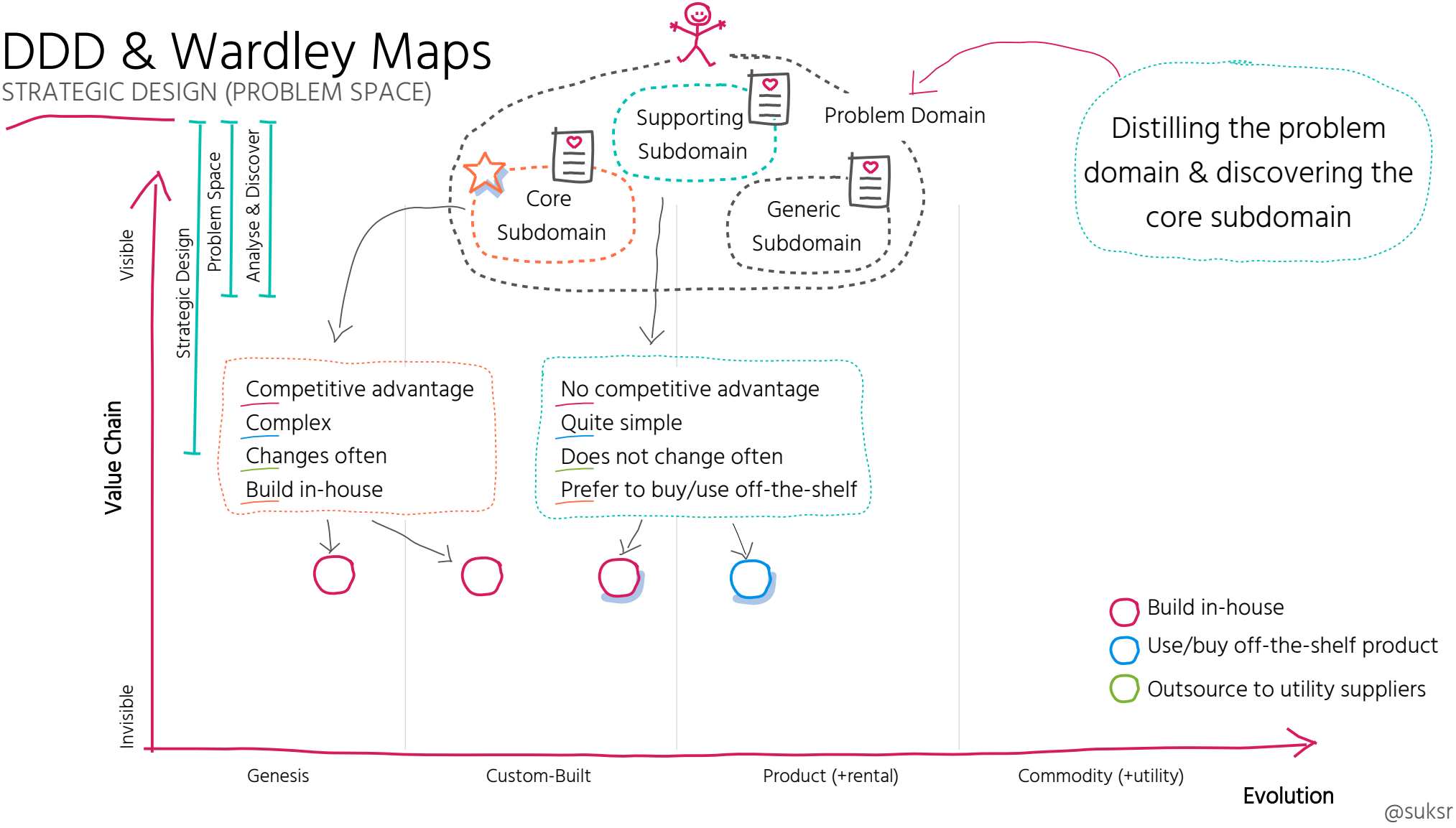
STRATEGIC DESIGN (PROBLEM SPACE)



@suksr

DDD & Wardley Maps

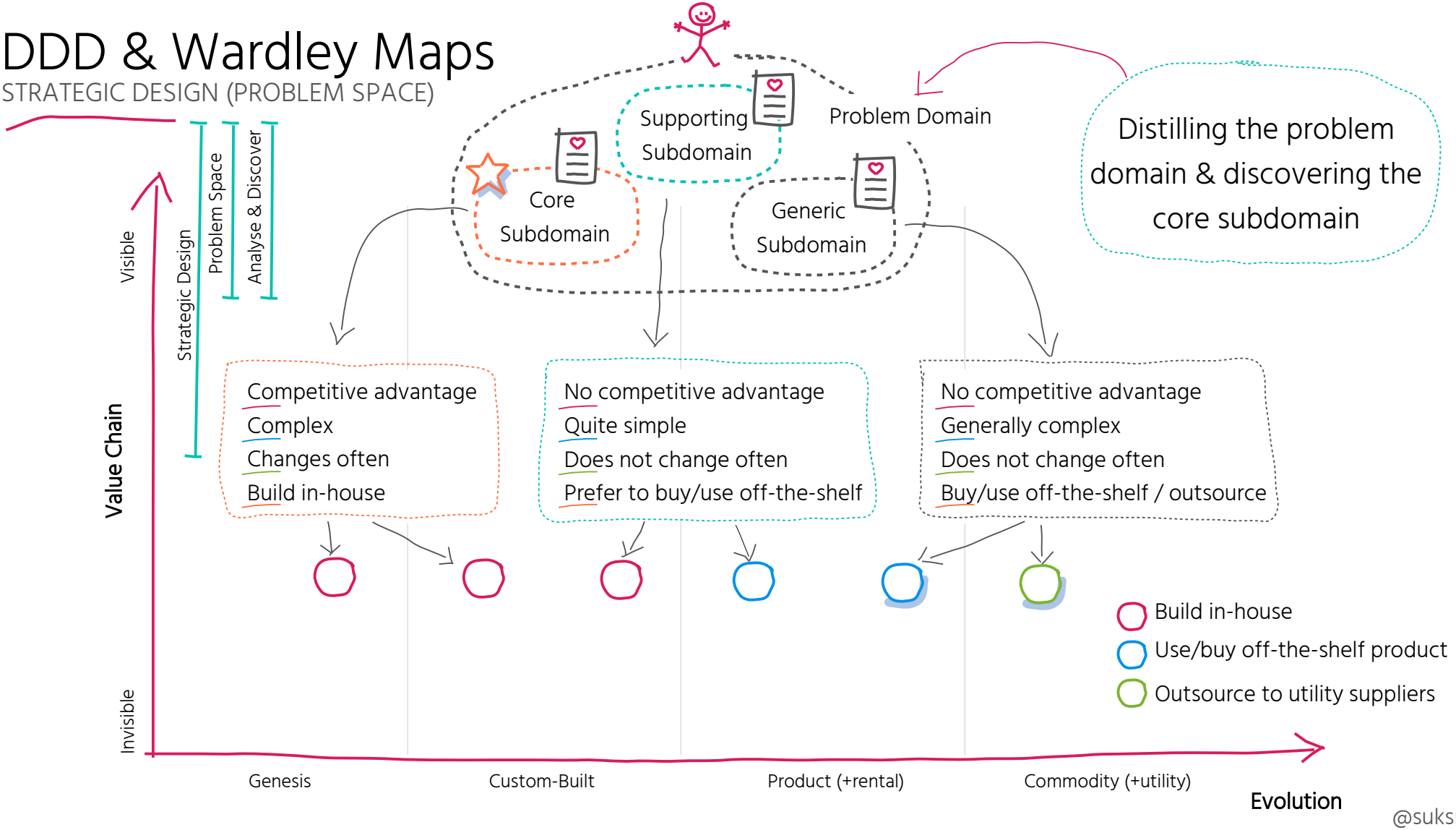
STRATEGIC DESIGN (PROBLEM SPACE)



@suksr

DDD & Wardley Maps

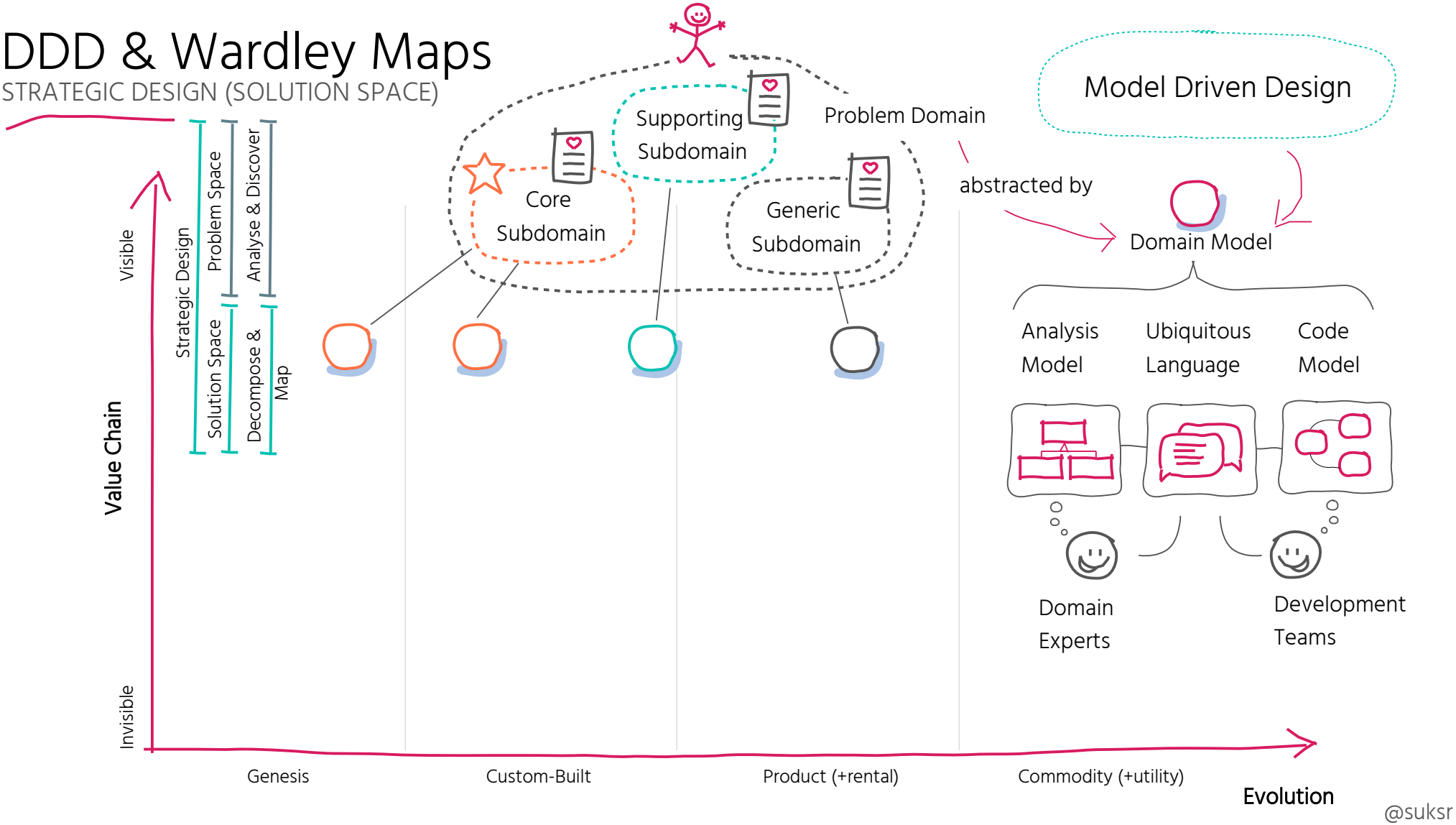
STRATEGIC DESIGN (PROBLEM SPACE)



@suksr

DDD & Wardley Maps

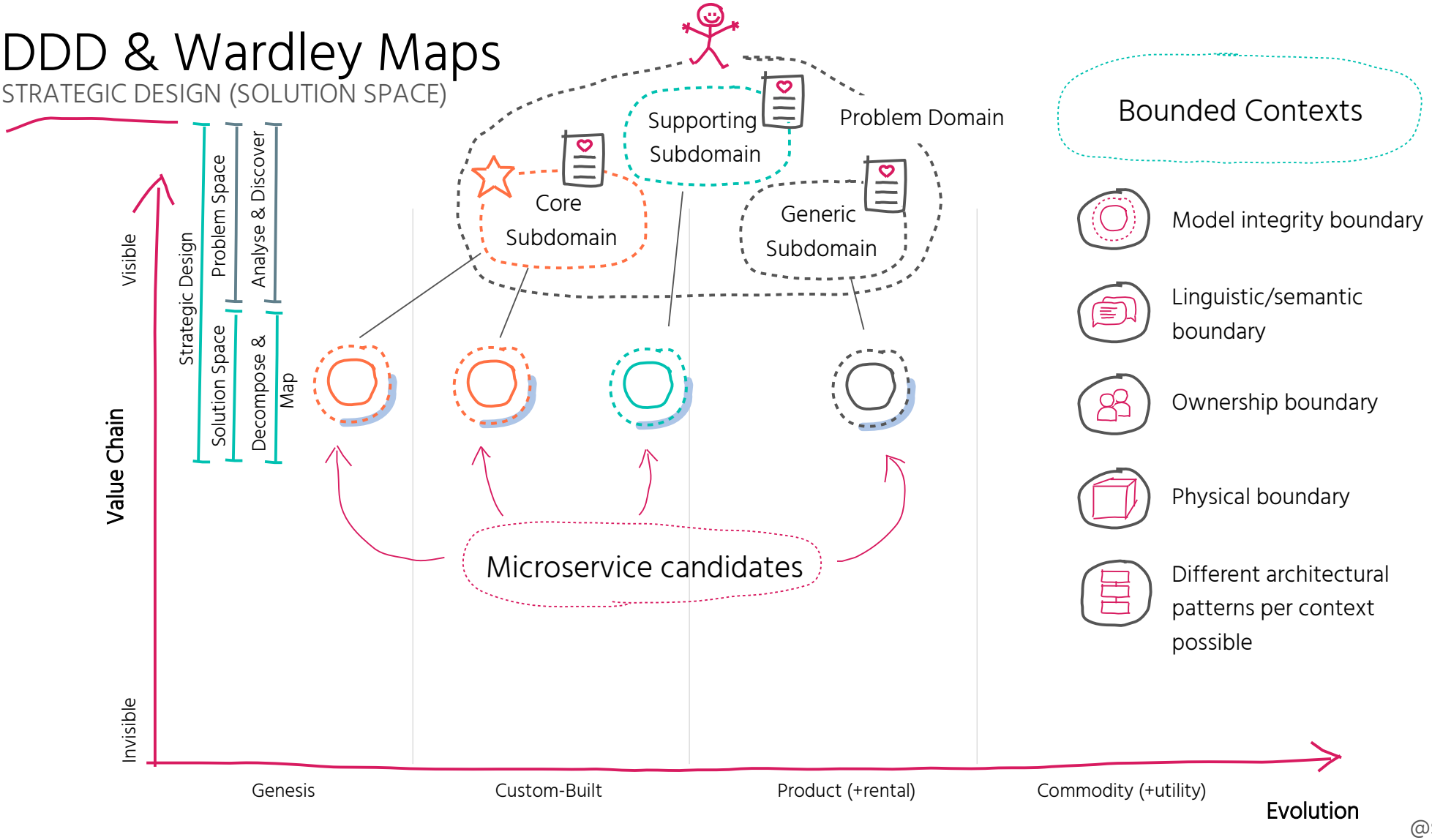
STRATEGIC DESIGN (SOLUTION SPACE)



@suksr

DDD & Wardley Maps

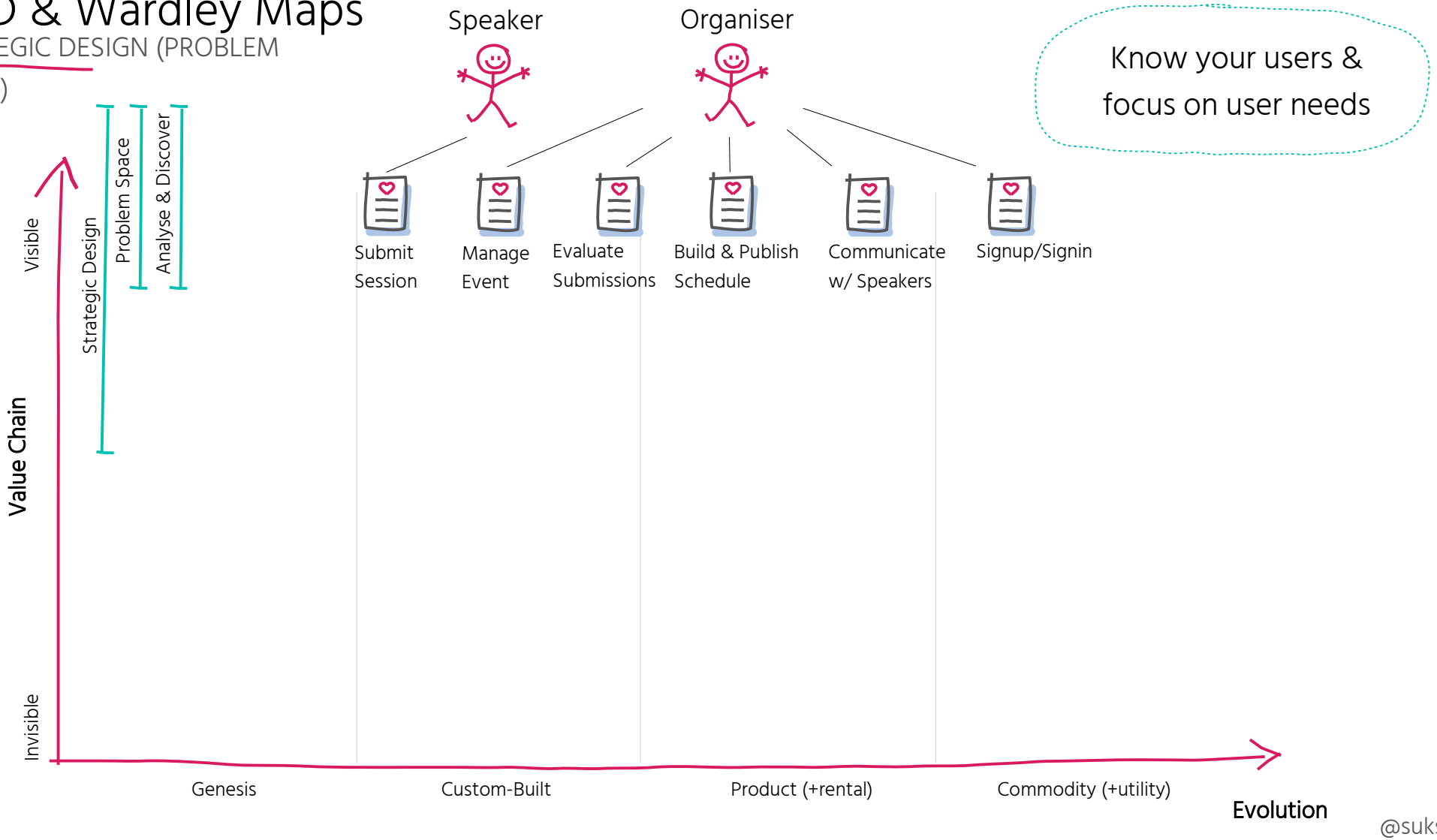
STRATEGIC DESIGN (SOLUTION SPACE)



@suksr

DDD & Wardley Maps

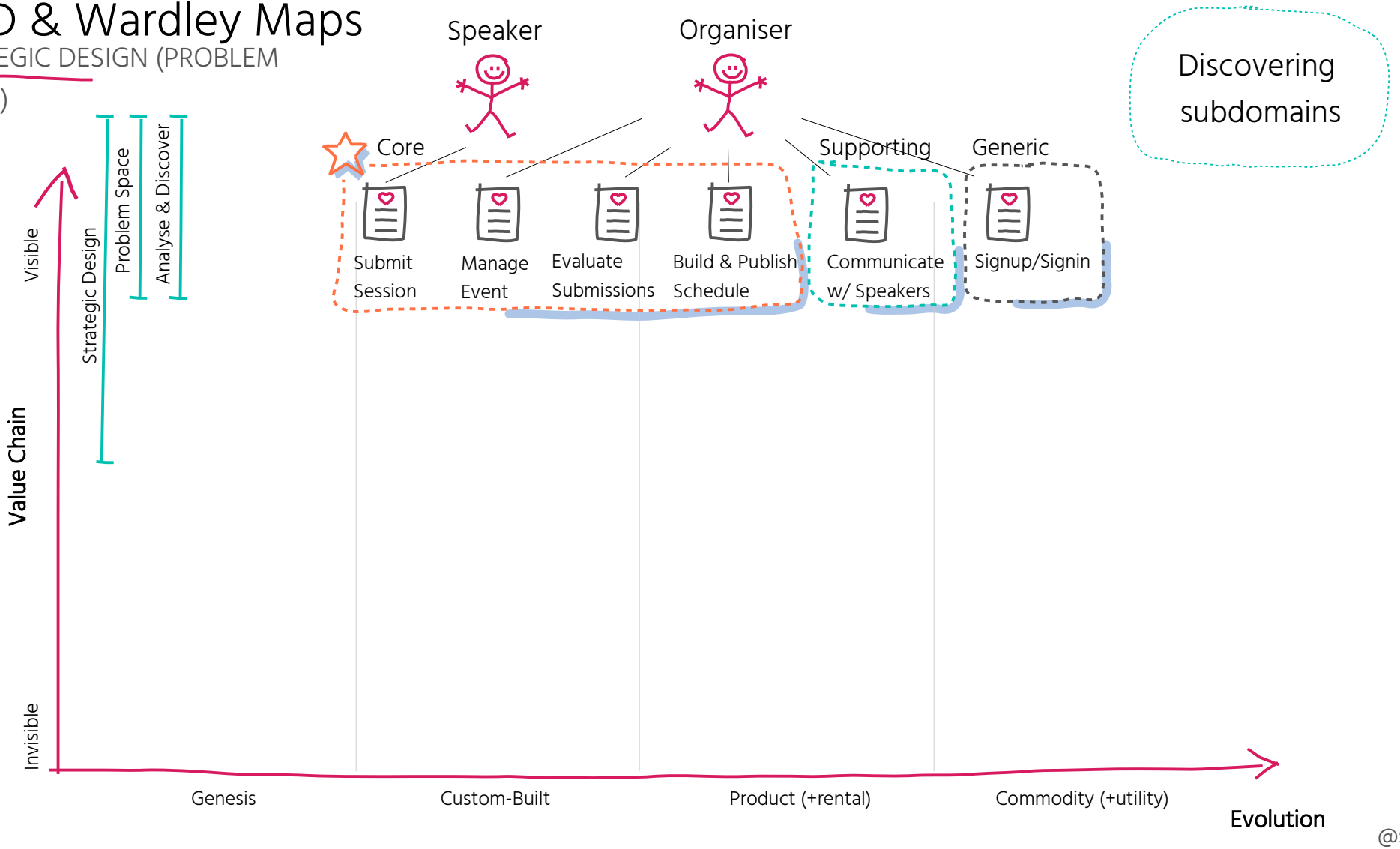
STRATEGIC DESIGN (PROBLEM SPACE)



@suksr

DDD & Wardley Maps

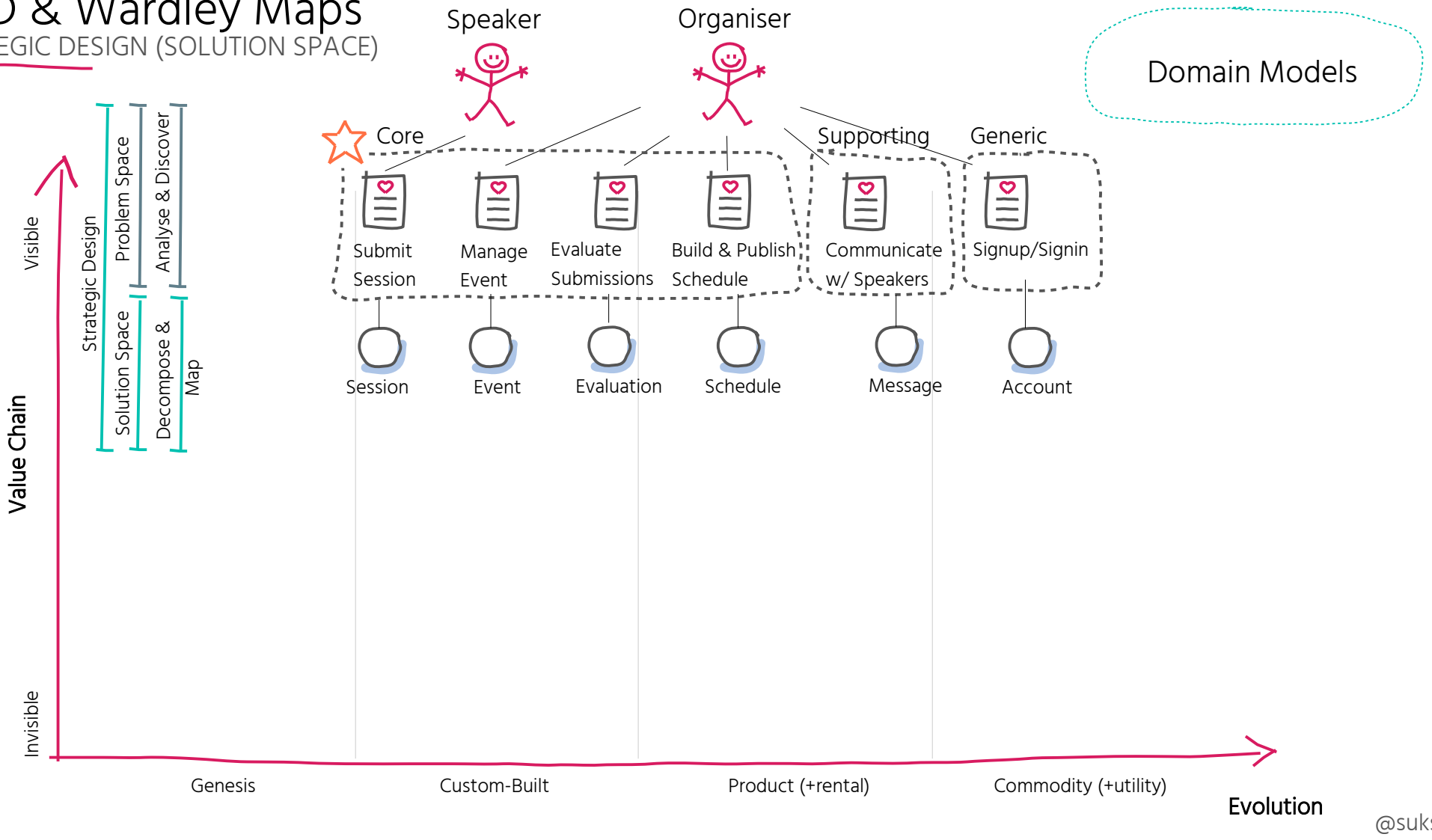
STRATEGIC DESIGN (PROBLEM SPACE)



@suksr

DDD & Wardley Maps

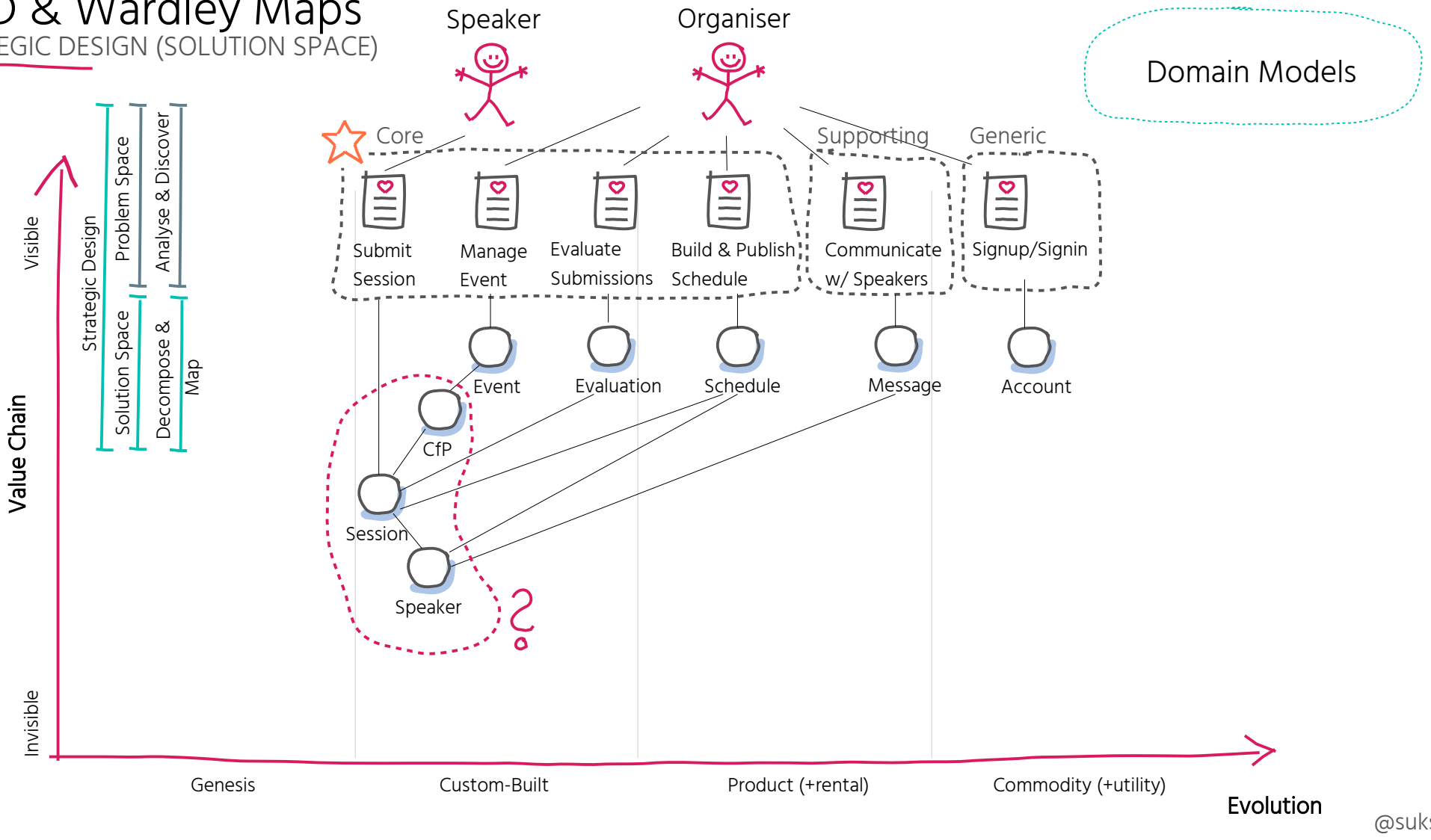
STRATEGIC DESIGN (SOLUTION SPACE)



@suksr

DDD & Wardley Maps

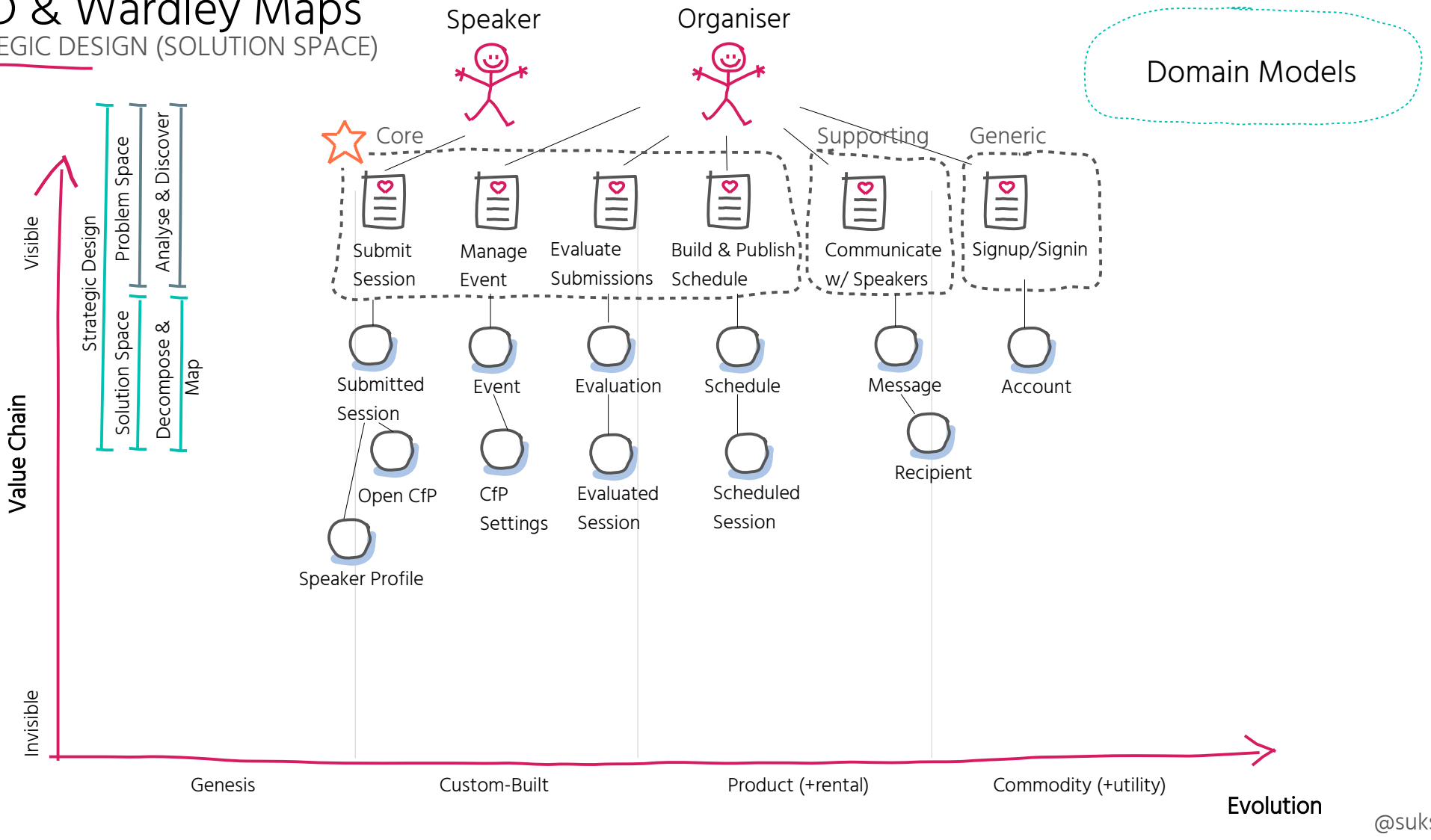
STRATEGIC DESIGN (SOLUTION SPACE)



@suksr

DDD & Wardley Maps

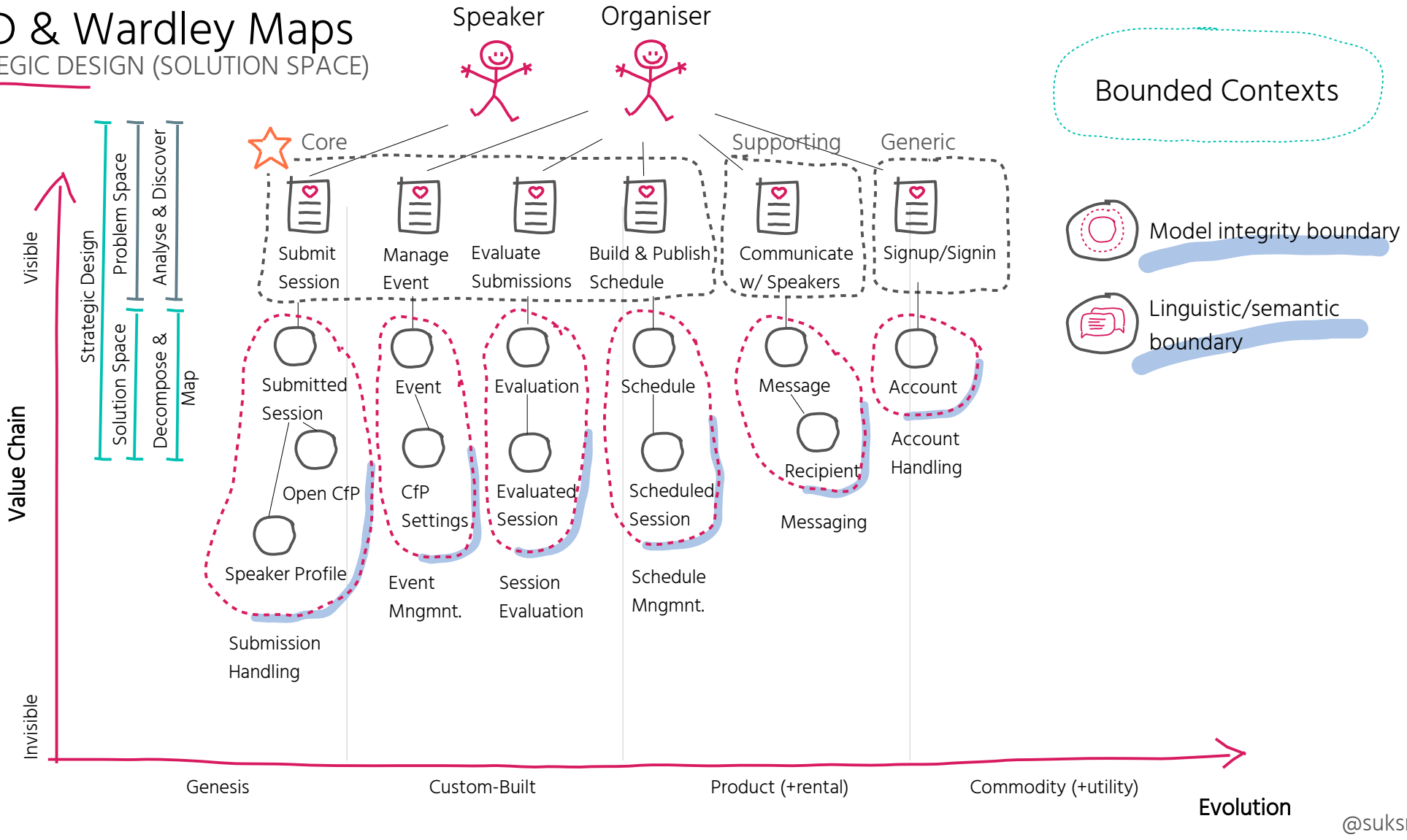
STRATEGIC DESIGN (SOLUTION SPACE)



@suksr

DDD & Wardley Maps

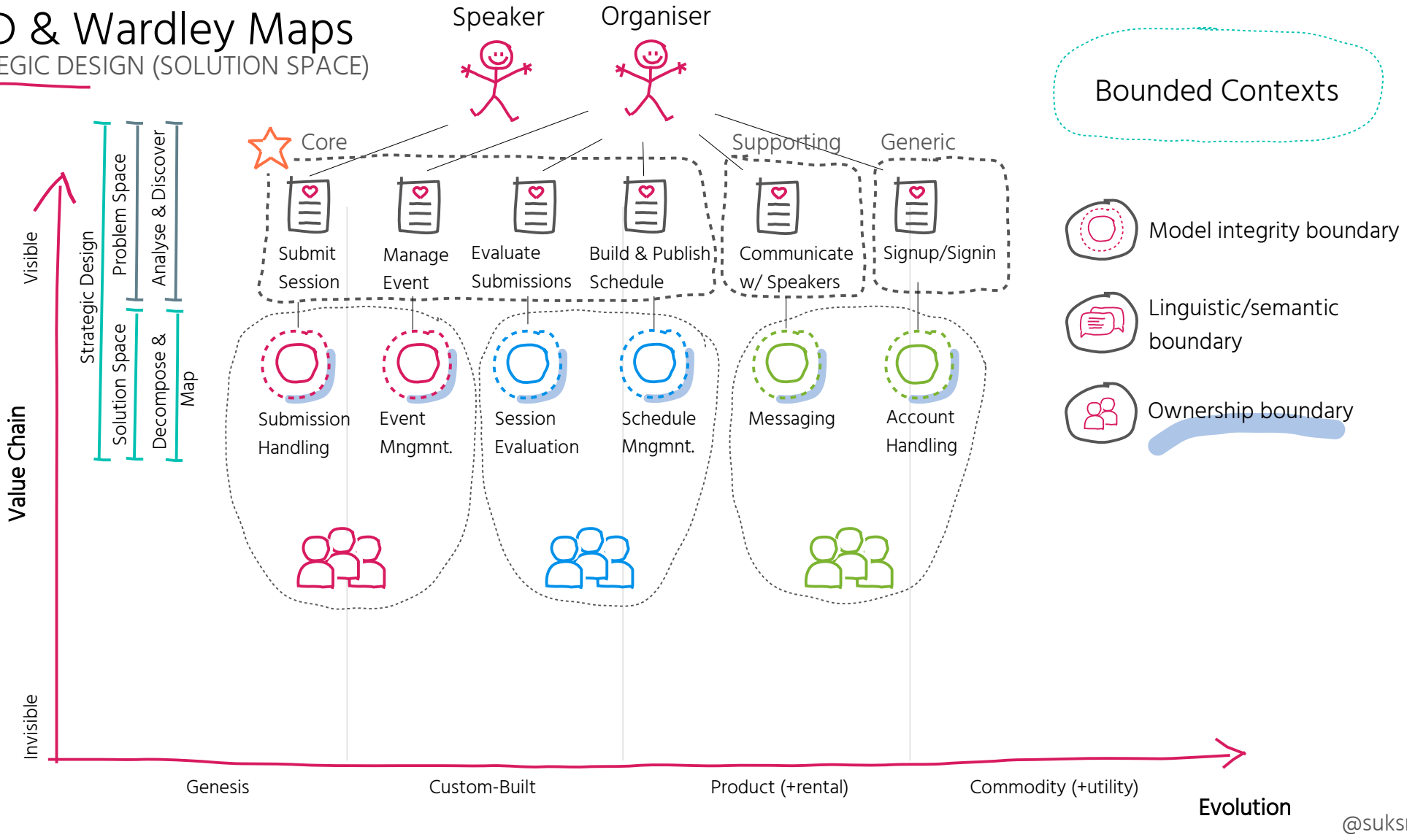
STRATEGIC DESIGN (SOLUTION SPACE)



@suksr

DDD & Wardley Maps

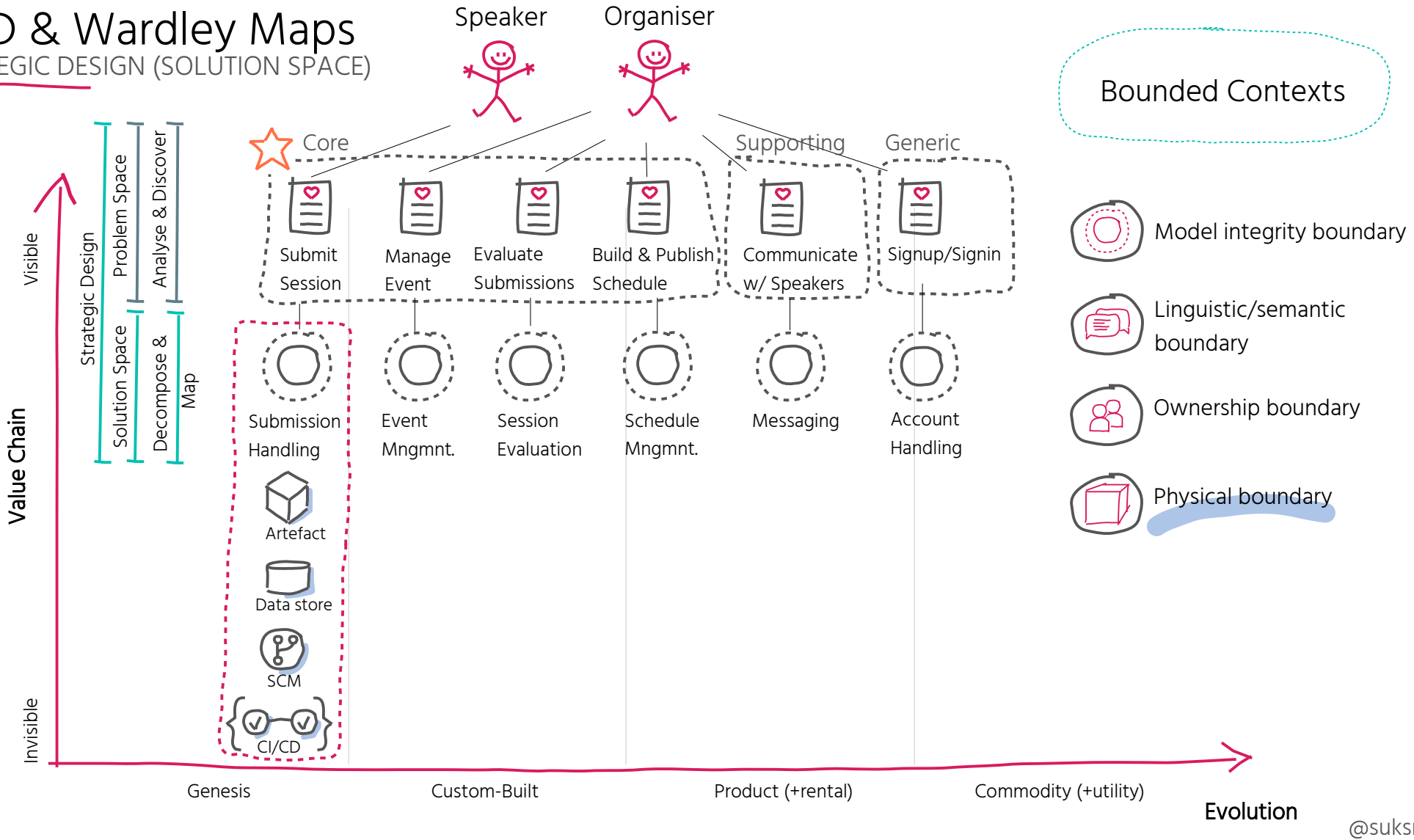
STRATEGIC DESIGN (SOLUTION SPACE)



@suksr

DDD & Wardley Maps

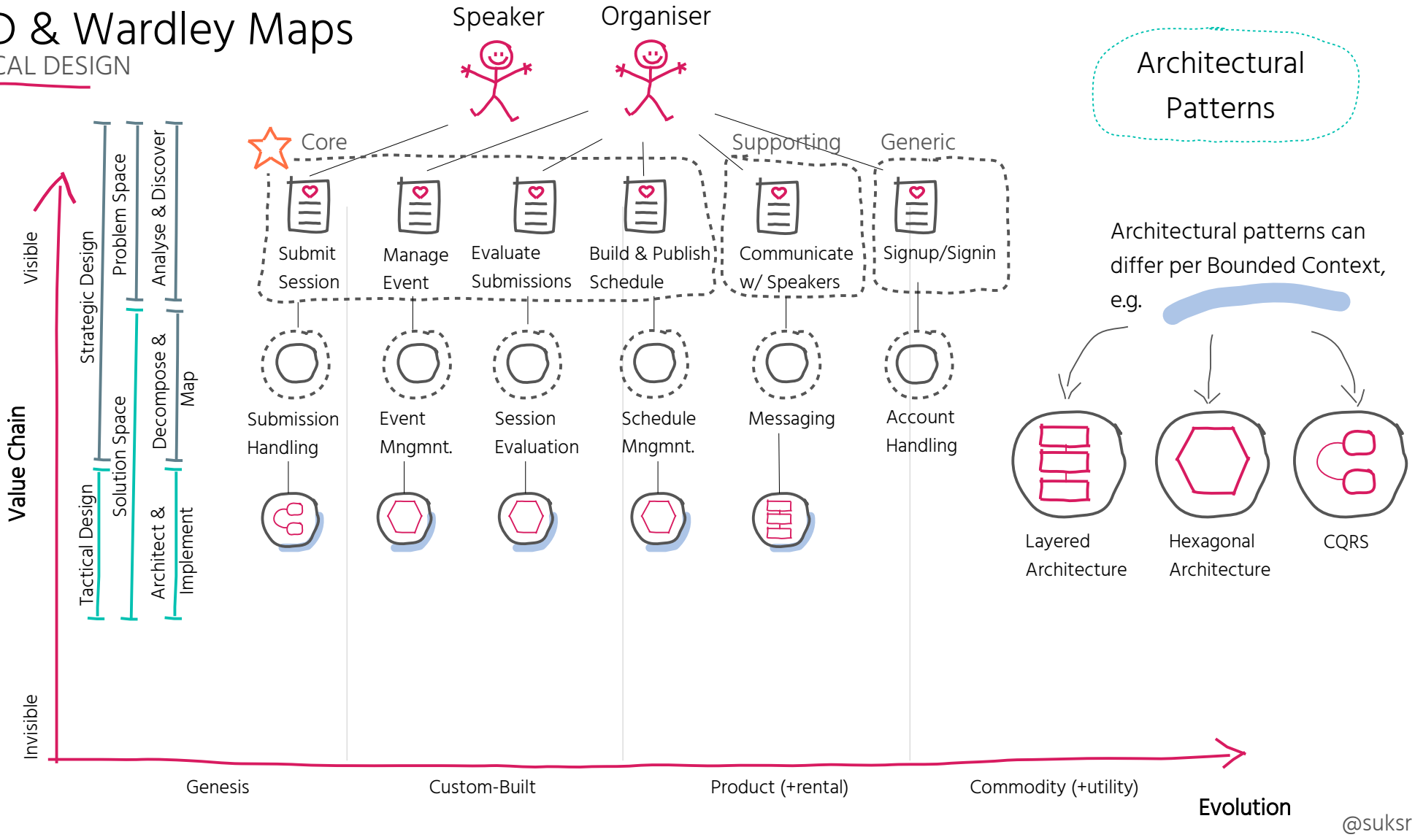
STRATEGIC DESIGN (SOLUTION SPACE)



@suksr

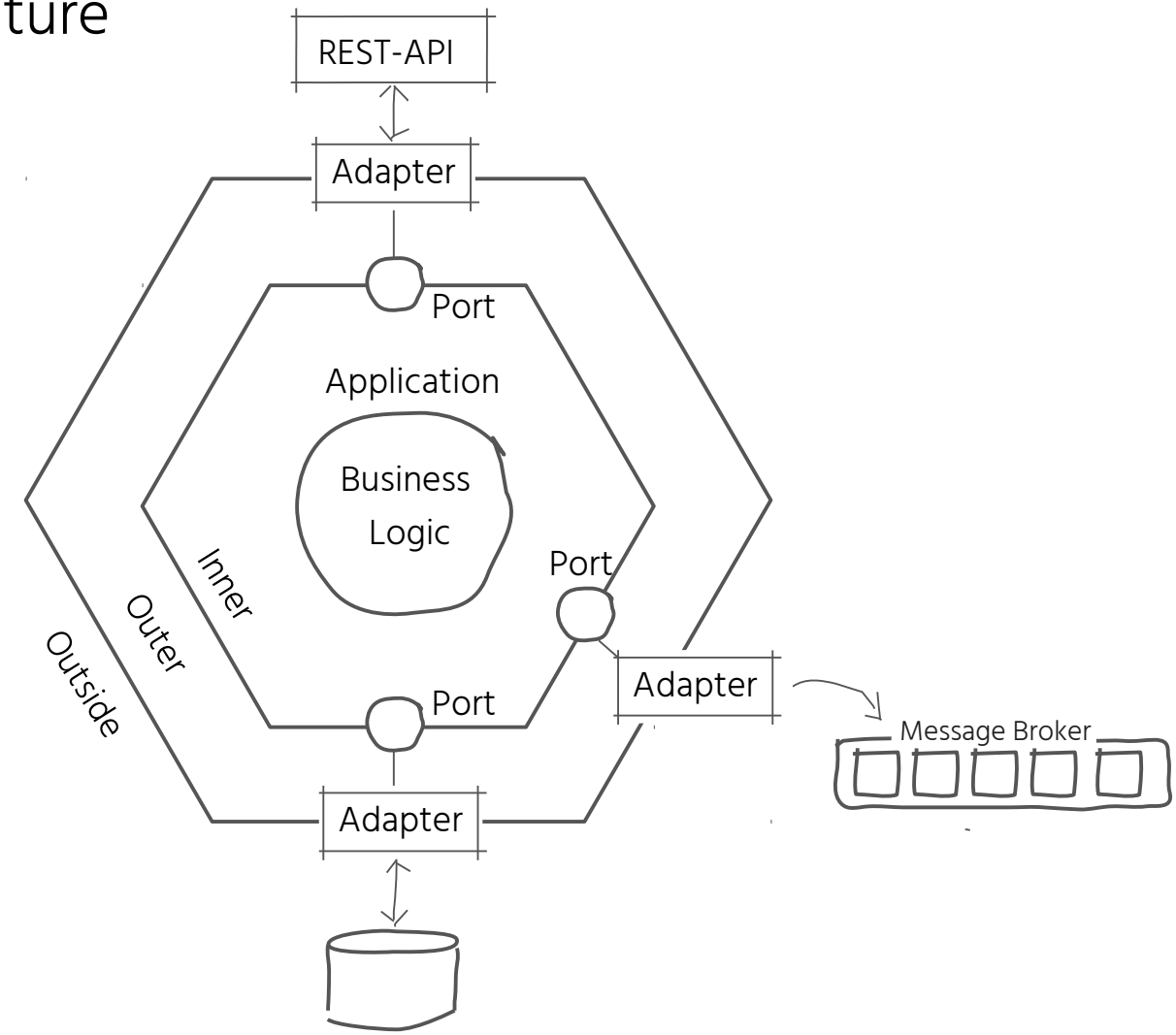
DDD & Wardley Maps

TACTICAL DESIGN



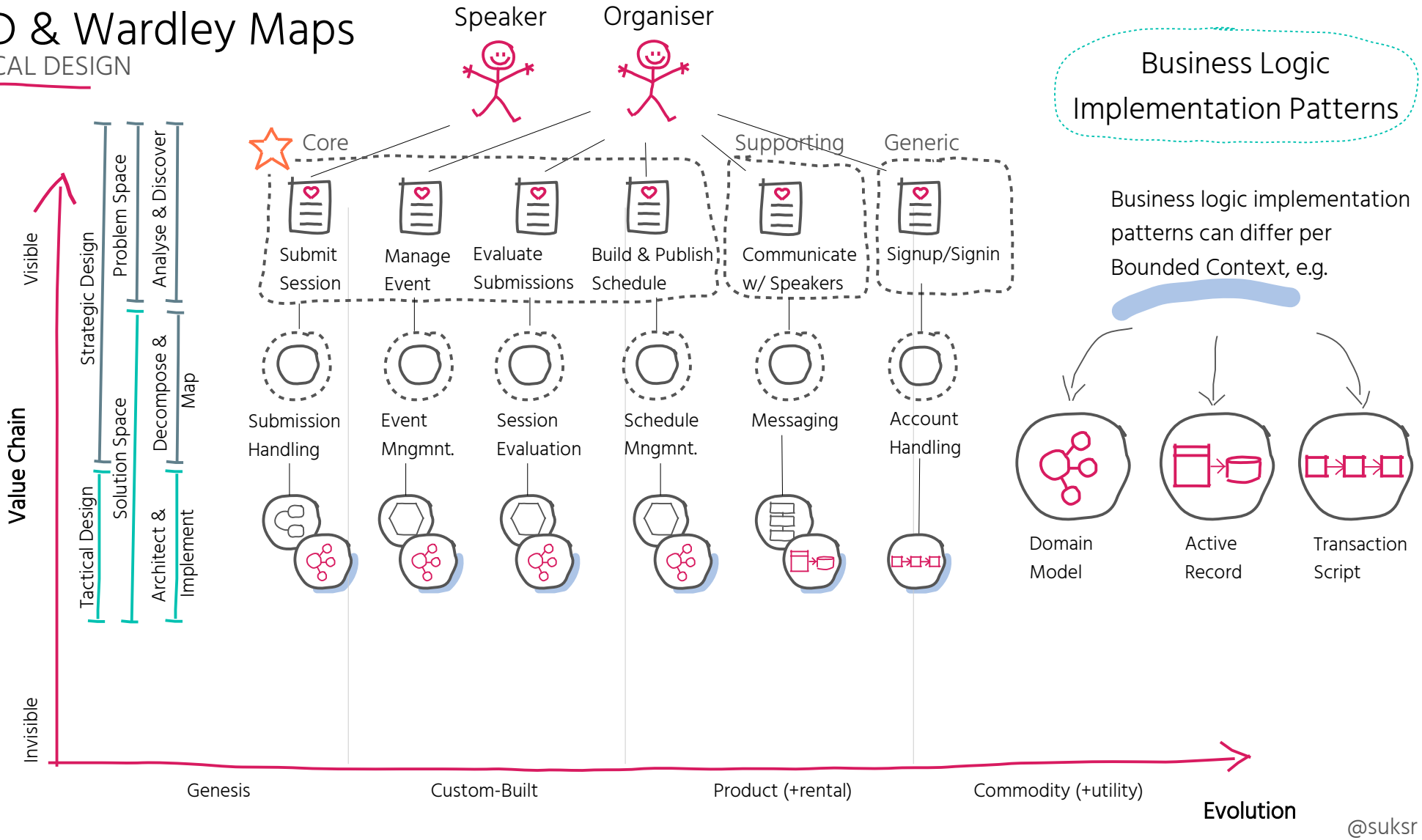
@suksr

Hexagonal Architecture



DDD & Wardley Maps

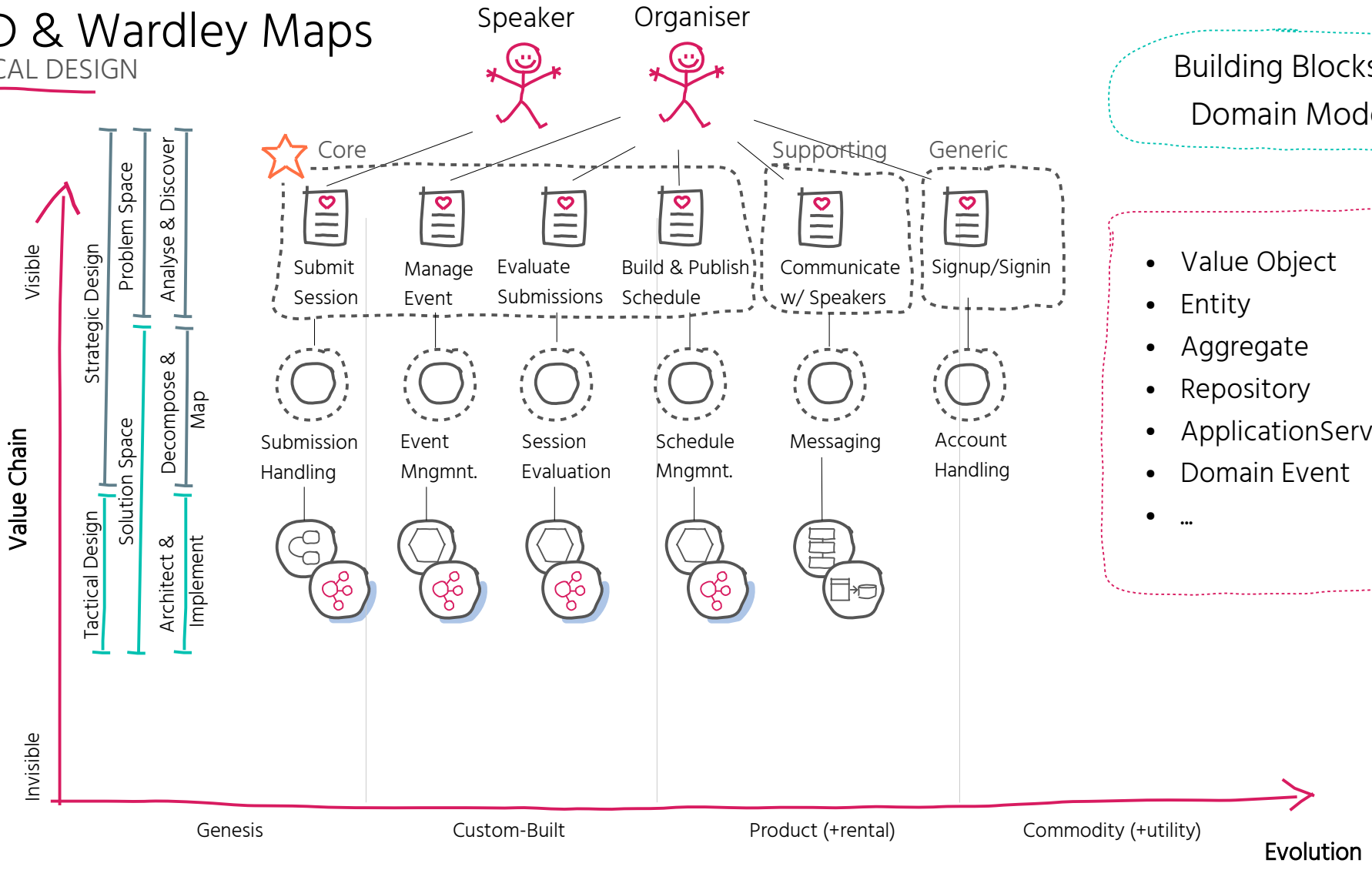
TACTICAL DESIGN



@suksr

DDD & Wardley Maps

TACTICAL DESIGN



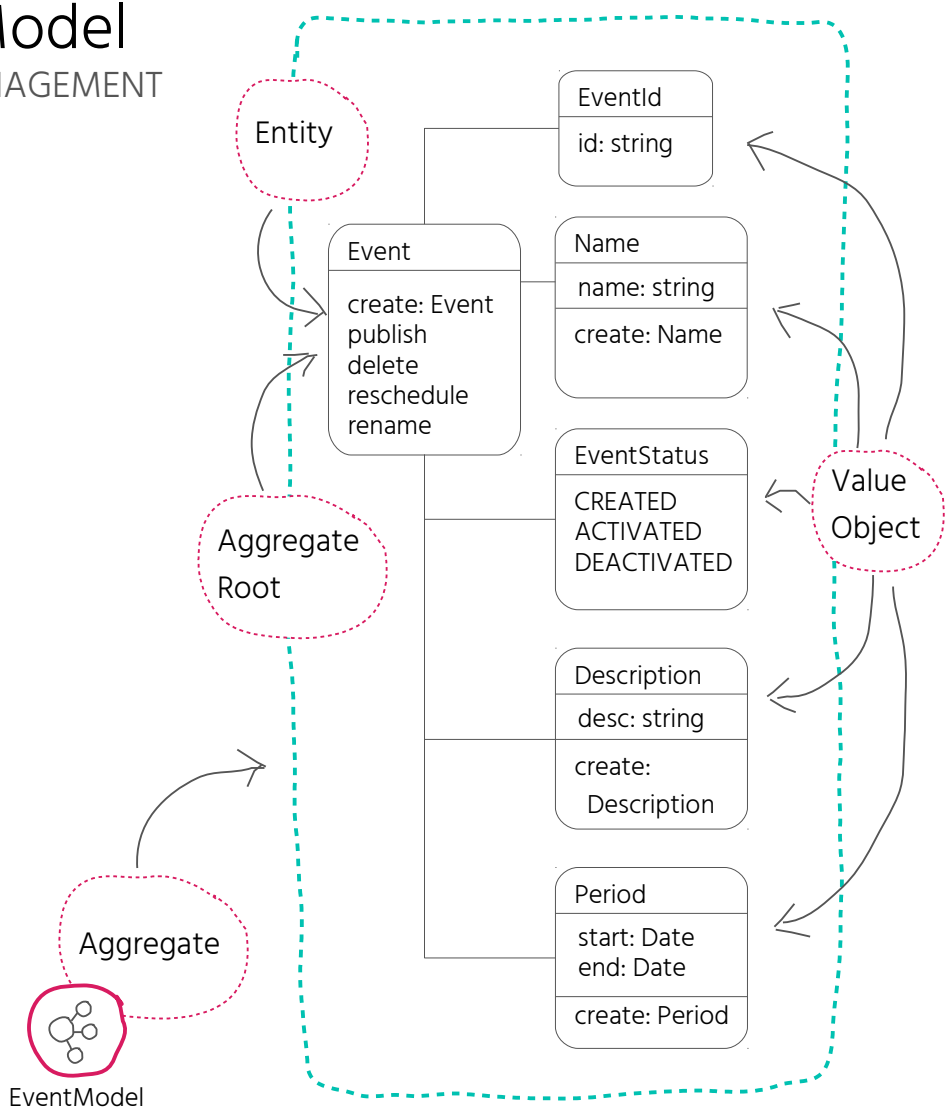
Building Blocks of Domain Models

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

@suksr

Example Domain Model

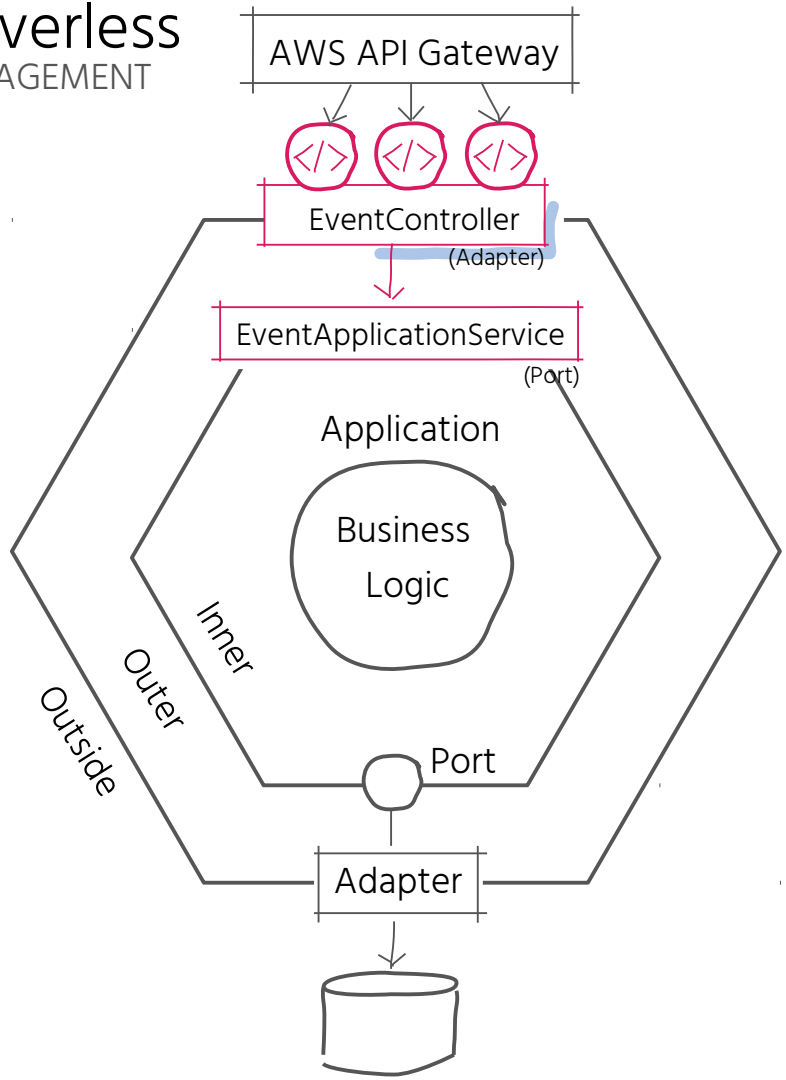
BOUNDED CONTEXT: EVENT MANAGEMENT



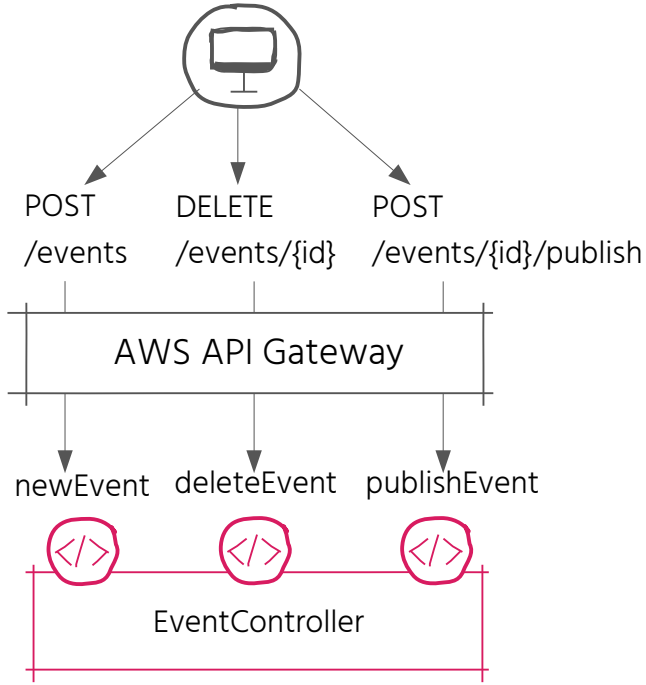
@suksr

Backend-API w/ Serverless

BOUNDED CONTEXT: EVENT MANAGEMENT



REST-API with AWS API-Gateway and AWS Lambda



@suksr

Backend-API w/ Serverless

BOUNDED CONTEXT: EVENT MANAGEMENT

REST-API Adapter

Port

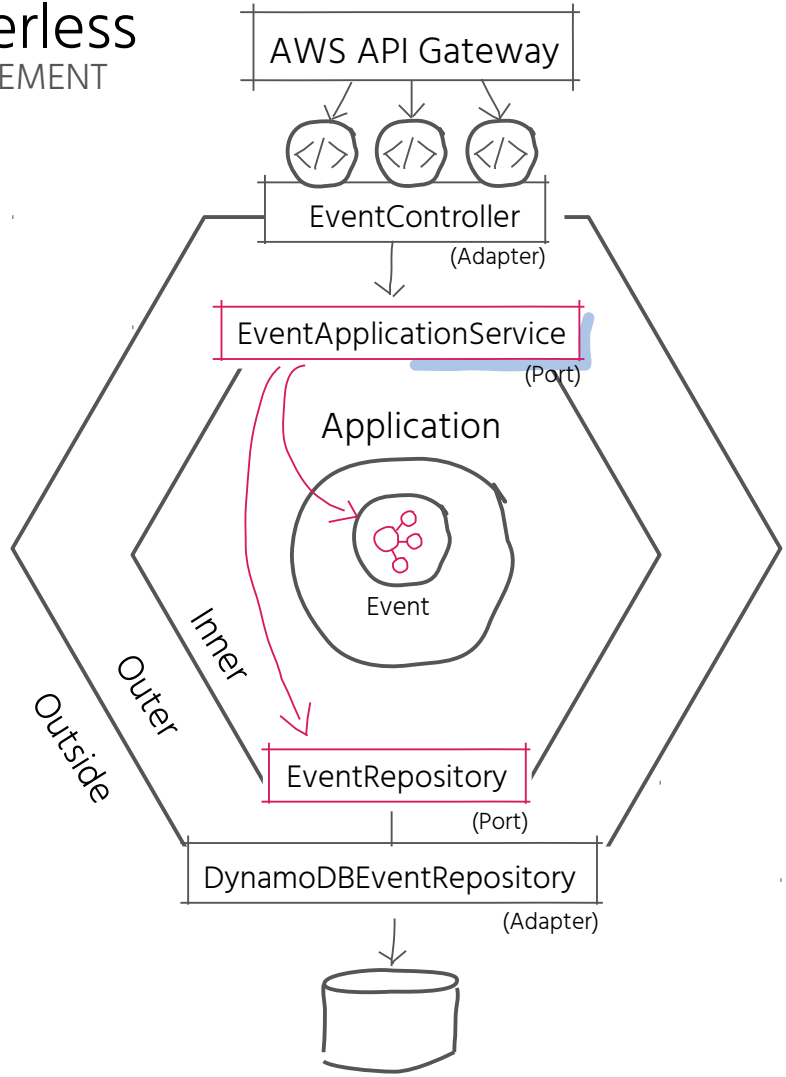
Lambda Function

Lambda Function

```
export class EventController {

    private readonly  eventService: EventApplicationService;

    public constructor(eventService: EventApplicationService) {
        this.eventService = eventService;
    }

    public publishEvent: Handler = async (event: APIGatewayEvent, context: Context, callback: Callback) => {

        if (!event.pathParameters && !event.pathParameters.id) {
            return callback(null, failure({ status: "error", error: "no event id specified" }));
        }
        const eventId = new EventId(event.pathParameters.id);

        try {

            await this.eventService.publishEvent(eventId);
            callback(null, success({ status: "ok" }));

        } catch(e) {
            return callback(null, failure({ status: "error", error: e }));
        }
    };

    public newEvent: Handler = async (event: APIGatewayEvent, context: Context, callback: Callback) => {

        // ... //

}
```

Backend-API w/ Serverless

BOUNDED CONTEXT: EVENT MANAGEMENT



Backend-API w/ Serverless

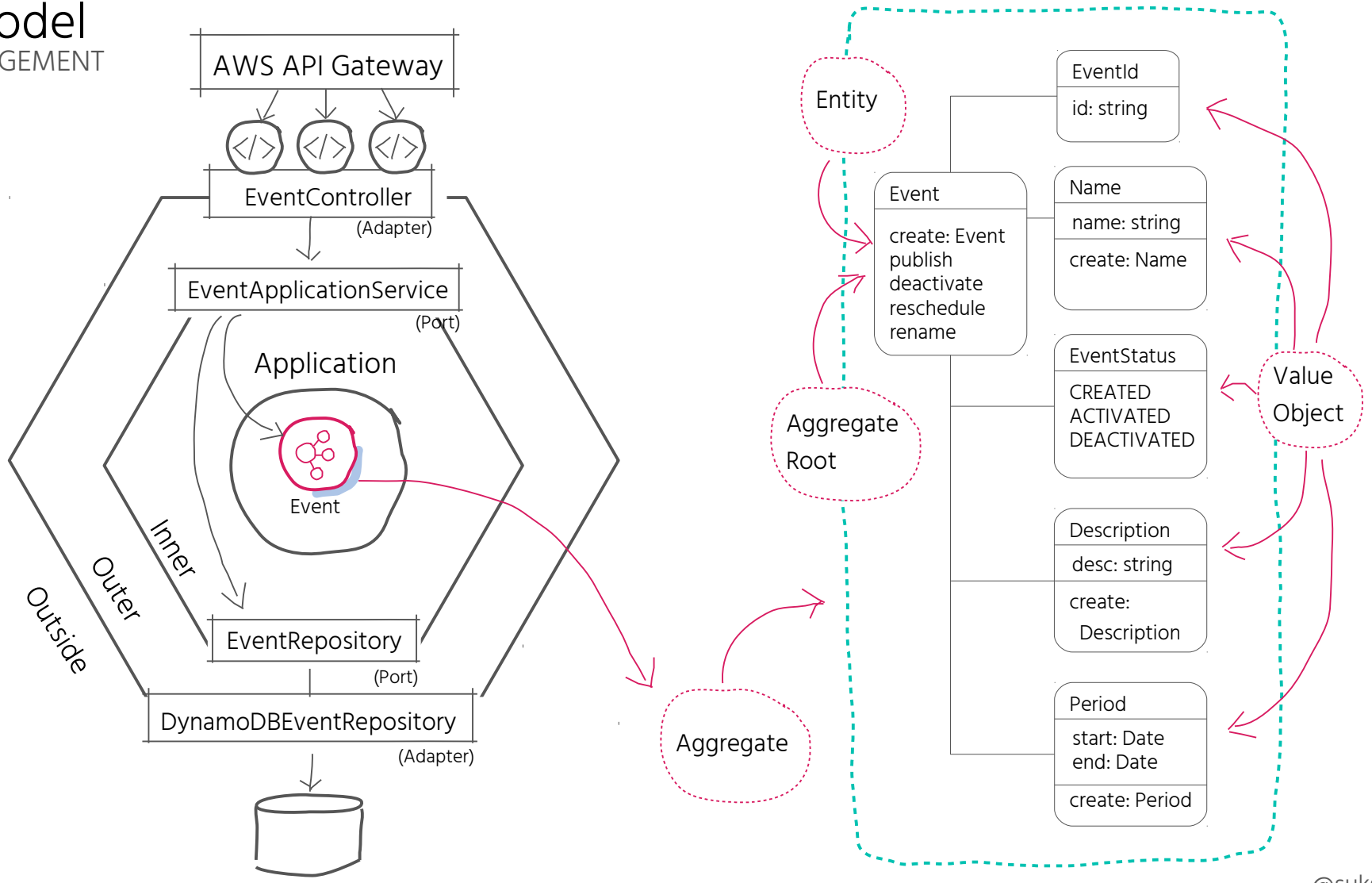
BOUNDED CONTEXT: EVENT MANAGEMENT

ApplicationService

```
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

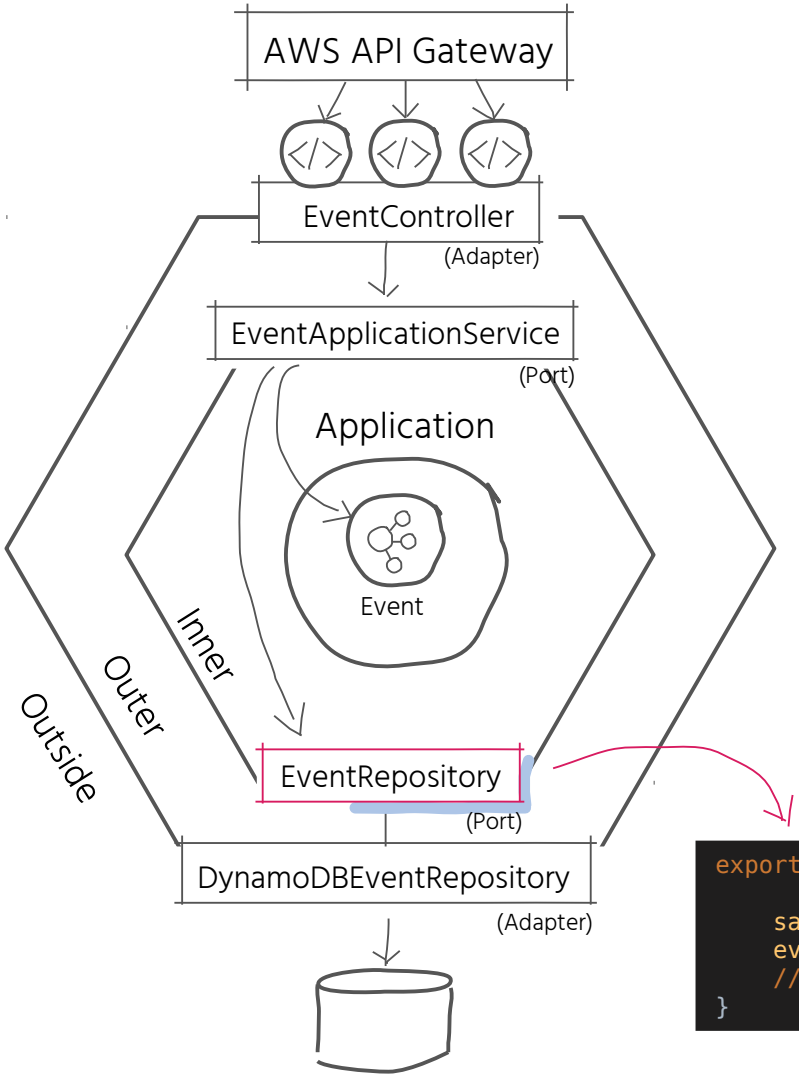
BC: EVENT MANAGEMENT



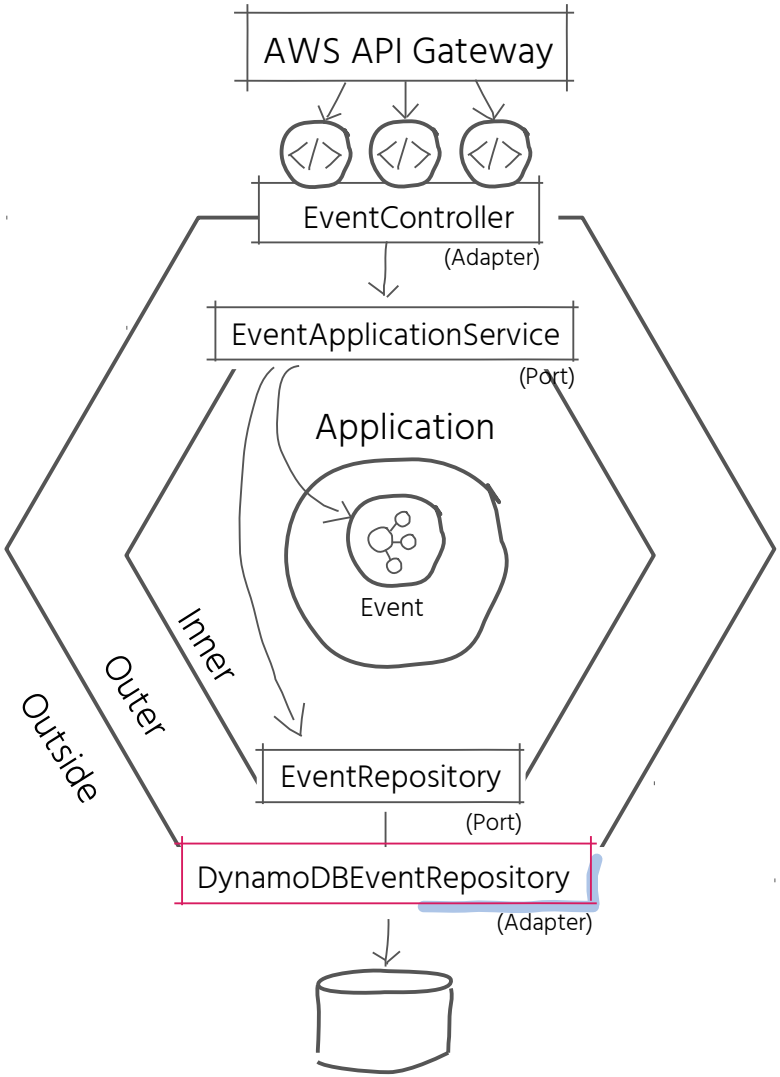
@suksr

```
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 {

        // ... //
}
```

Aggregate




```
export default interface EventRepository {

    saveEvent(event: Event): void;
    eventOfId(id: EventId): Promise<Event|null>;
    // ... //
}
```

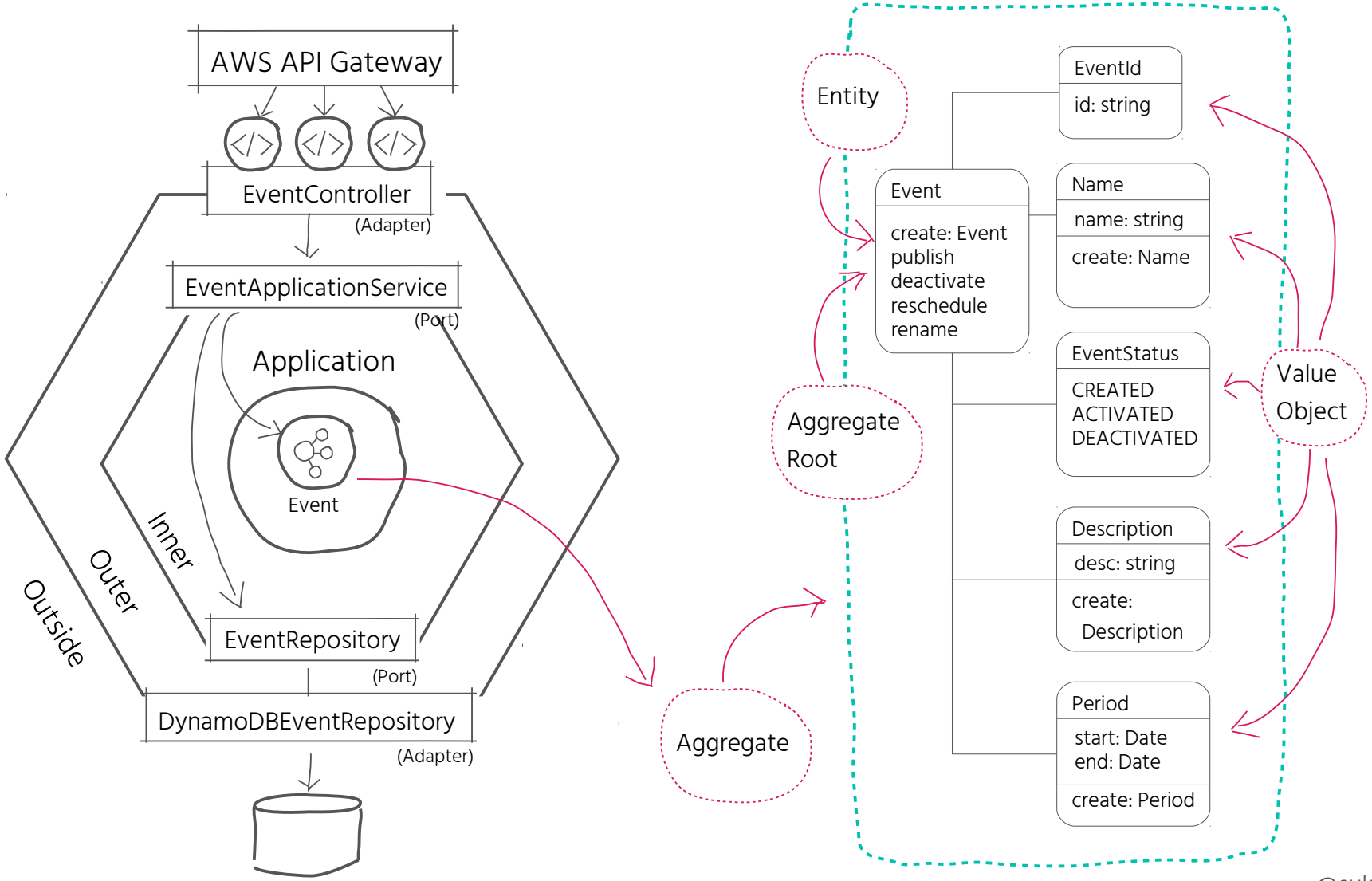


Data Storage with
AWS DynamoDB

Database Adapter

```
export default class DynamoDBEventRepository implements EventRepository {

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

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

    public saveEvent(event: Event) {
        const params : DocumentClient.PutItemInput = {
            TableName: DynamoDBEventRepository.TABLE_NAME,
            Item: {
                eventId: event.id.toString(),
                name: event.name.value,
                startDate: event.period.startDate.toISOString(),
                endDate: event.period.endDate.toISOString(),
                description: event.description ? event.description.value: undefined,
                eventStatus: event.status,
                cfp: event.cfp ? {
                    description: event.cfp.description.value,
                    startDate: event.cfp.period.startDate.toISOString(),
                    endDate: event.cfp.period.endDate.toISOString(),
                    id: event.cfp.id ? event.cfp.id.toString(): null
                }: null
            }
        };

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

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

        // ... //
}
```

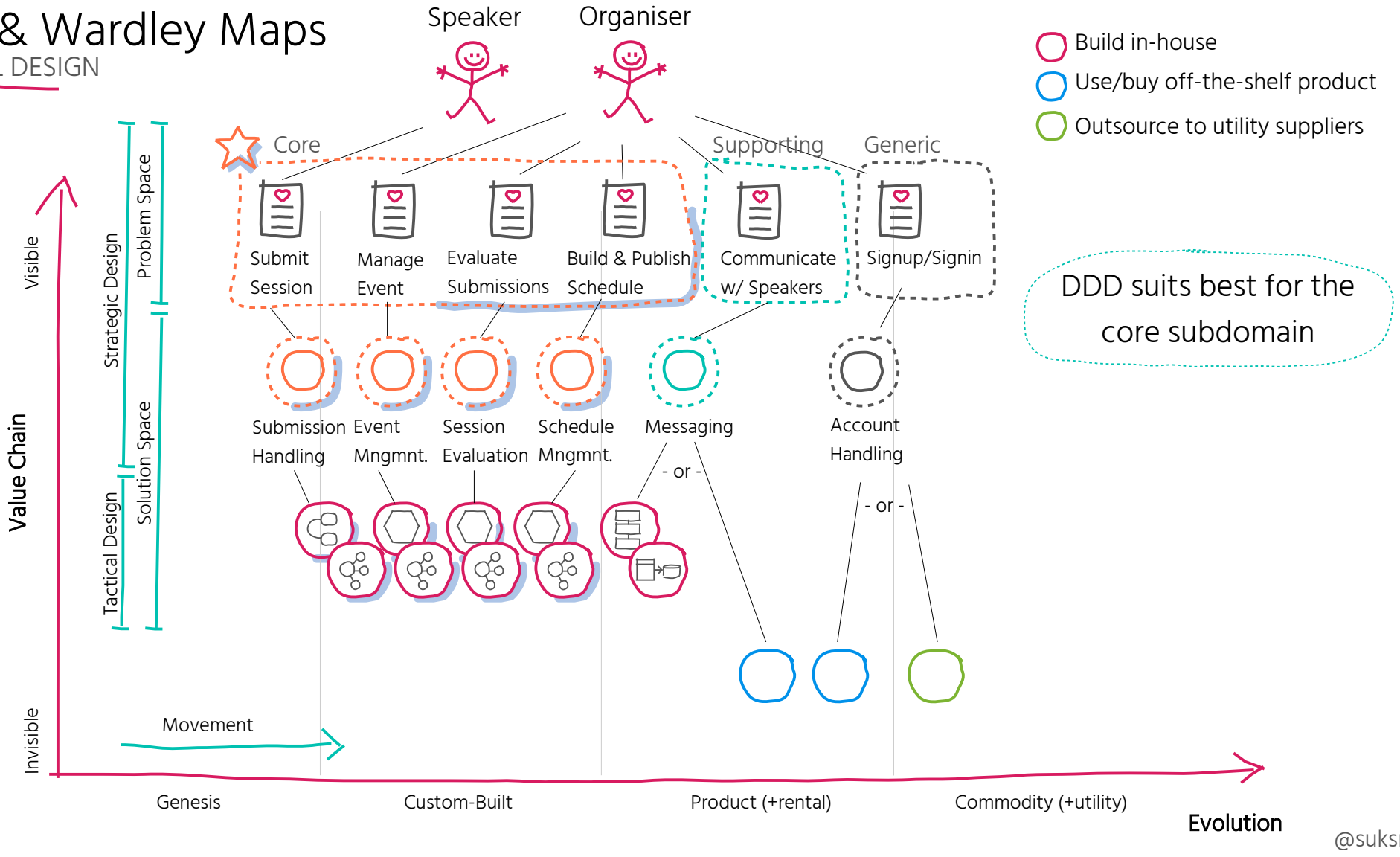
@suksr


AWS API Gateway
EventController
(Adapter)
EventApplicationService
(Port)
Application
Event
Inner
Outer
Outside
EventRepository
(Port)
DynamoDBEventRepository
(Adapter)
Aggregate
Entity
Aggregate Root
Value Object
EventId
id: string
Event
create: Event
publish
deactivate
reschedule
rename
Name
name: string
create: Name
EventStatus
CREATED
ACTIVATED
DEACTIVATED
Description
desc: string
create: Description
Period
start: Date
end: Date
create: Period

@suksr

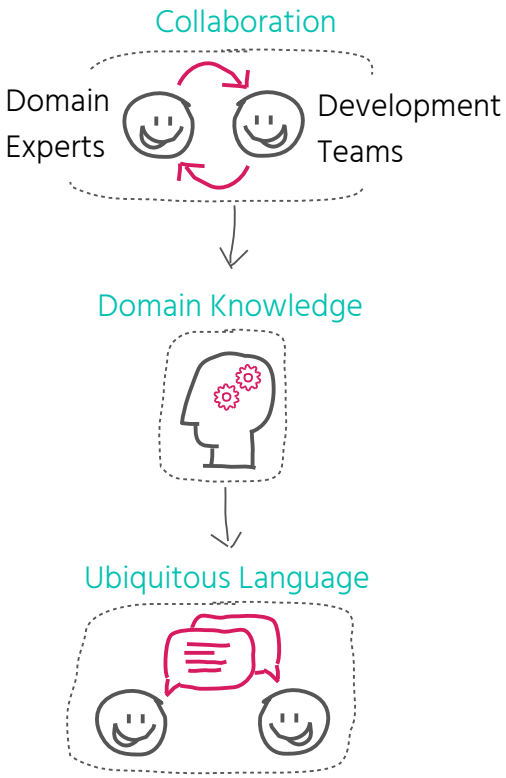
DDD & Wardley Maps

TACTICAL DESIGN

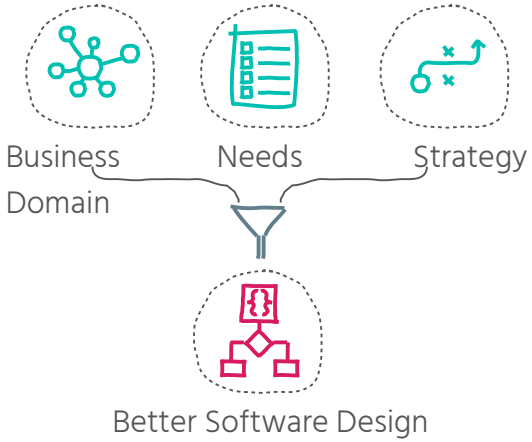


@suksr

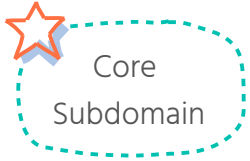
DDD helps with …



Gaining Domain Knowledge



Aligning Software Design to Business Domain

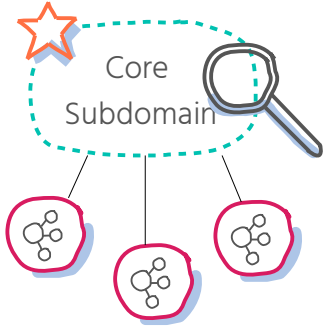


Discovering the Core Subdomain

Decomposing into Modular Components

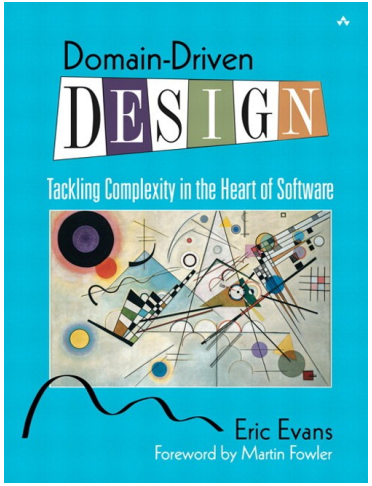
Do not apply DDD everywhere!

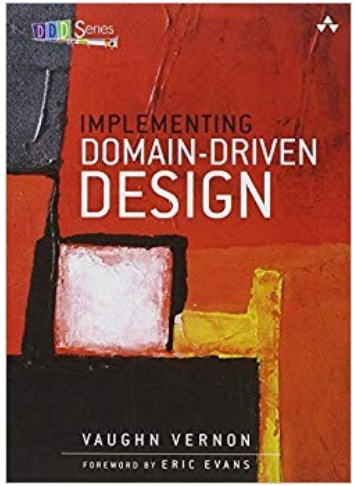
Focus on your core!

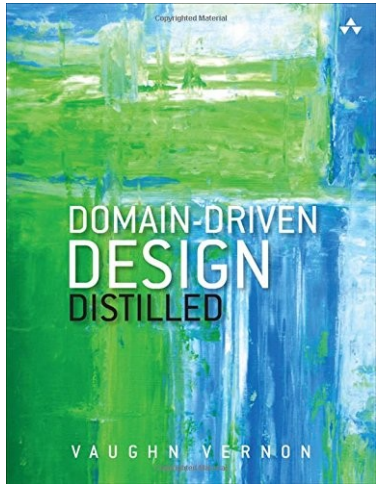


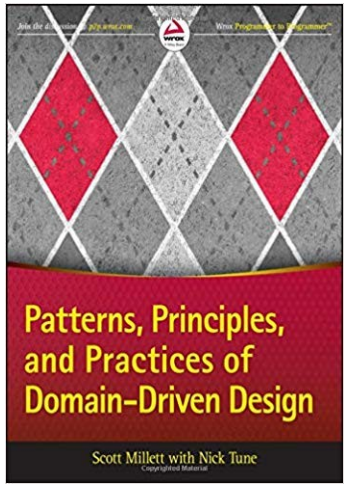
@suksr

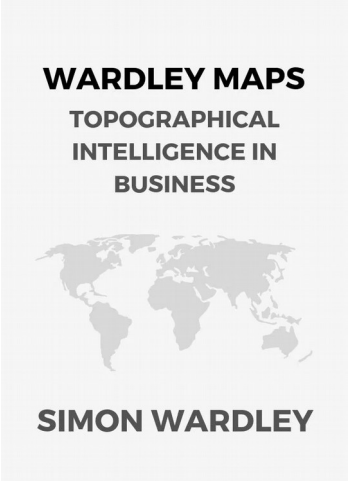
Some References











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