Content

• Introduction
• Divergence in Software-Systems
• Complexity in Software-Systems
• DSE Fundamentals
• Alignment & Continuous Integration
• Consequences for Industrial Software Development
• Conclusions
• References
DSE

Introduction
Domain Software Engineering [DSE] = an architectural *methodology* for evolving a *software* system that closely aligns to *business* domains
There are different ways to approach software development.

For the last 20 years, the software industry has known and used many methods to create its products – each with its advantages and shortcomings.

Here we investigate the method which started as «Domain-Driven Design».
1990 ... today:
Various Software Development Methodologies

2004:

Download:
http://www.infoq.com/minibooks/domain-driven-design-quickly
[last accessed: 27.1.2016]
DSE Context

New Business Requirements → Project → Development Process

Existing SW-System → Divergence → Complexity

Evolved SW-System

Prof. Frank J. Furrer - WS 2015/16
Divergence

Development Process

Complexity

Serious obstructions introduced by many software development processes
DSE

Divergence
Divergence =
Mismatch between Business Needs and IT-Implementation
Example: «Please build a swing for my little daughter»
Requirements

How the customer explained it

How the business consultant described it

Specifications

How the project leader understood it
Implementation

How the analyst designed it

How the programmers implemented it

How the project was documented
Deployment → Operation

When the project was delivered → What the customer really wanted

http://projectcartoon.com/create/
What is the reason?

Failed Communications!

• Different *vocabulary* between business and IT
• Lots of *implicit* knowledge and assumptions
• No common *model*
Ringvorlesung 2016: Domain Software Engineering

DSE

Complexity
"Complexity is that property of an IT-system which makes it difficult to formulate its overall behaviour, even when given complete information about its parts and their relationships."

\[ \text{Complexity} = (\text{IT-}) \text{ Risk} \]
2 types of complexity

**Essential** complexity

**Accidental** Complexity

http://www.acordceo.org

https://acoarecovery.wordpress.com
**Essential** complexity

... is the *inherent* complexity of the system to be built. Essential complexity for a given problem *cannot* be reduced. It can only be lessened by *simplifying* the requirements for the system extension.

**Accidental** Complexity

... is *introduced* by our development activities or by constraints from our environment. This is unnecessary and can be *reduced* or eliminated.

⇒ Development methodology!

**Combat** accidental complexity

Manage essential complexity

http://www.sherweb.com
Domain Logic Complexity

Software Complexity

Legacy code base complexity

deliberate
accidental

Complexity form technical solution

blurred lines
DSE

Fundamentals (1/2)
Divergence = Mismatch between: Business Needs ⇔ IT-Implementation

Frustration!

DSE

Essential Complexity

Accidental Complexity

Misunderstandings
Lack of Precision
Semantic Differences

Customer/Business Needs
IT Implementation

https://cimx.wordpress.com

http://clipartzebra.com
Which are the key elements of DSE (Domain Software Engineering)?

1. Understanding the Business/Application Domain in terms of the business (⇒ Domain Model)
2. Use of an ubiquitous language (Business ↔ IT alignment)
The DSE concepts:

Business/Application Domain

Bounded Context

Domain Model

Anticorruption Layer
Business/Application Domain =

A Domain is a Sphere of Knowledge, Influence or Activity.

A Domain lives within a Bounded Context.

A Domain represents a well-defined Part of the Real World.

A Domain encapsulates a Domain Model.
Bounded Context =

The Bounded Context is the Boundary of a Model.

When you have multiple Models you should define Bounded Contexts.

To map between Bounded Contexts you use a Context Map.
Domain Model =

A Domain Model is a representation of the Entities, Relationships and their Properties in your Domain

The Domain Model should be recognizable and understandable by the business and IT

The domain model has sufficient essential details
Anticorruption Layer =

An Anti-Corruption Layer is a method to isolate two domains or systems, allowing systems to be integrated without knowledge of each other.

An Anti-Corruption Layer presents a Facade to both systems, defined in terms of their specific models.

Anti-Corruption Layers maintain the integrity of differing systems and models.
Definitions: Summary

- **Bounded Context A**
- **Bounded Context B**

**Business/Application Domain A**

**Domain Model A**

**IT Implementation A**

**Domain B**

**Anticorruption Layer**

Transformation
Examples:

Business/Application Domain

Bounded Context

Domain Model

Anticorruption Layer
Example: Business/Application Domain

Domain = Flight Monitoring

Context:
Thousands of planes are in the air all over the planet. The flight monitoring systems track every flight and avoid collisions.
Example: Bounded Context ⇔ SKYGUIDE Switzerland

Boundary = Contractual Responsibility within the European System
Example: Bounded Context

Anticorruption Layer = X-Compatibility Layer
Domain Model =

Reminder: A Domain Model is a representation of the Entities, Relationships and their Properties in your Domain
Dialog

Domain Expert

IT Expert

Business Domain

account

Checking account

mortgage account

Customer

Domain Model

account

customer

Checking account

mortgage account

owns

owns

owns
«A domain model is not just the knowledge in a domain expert’s head - … it is a rigorously organized and selective abstraction of that knowledge»

Eric Evans, 2004
Example 1: Flight Monitoring Domain Model

... Development of the Domain Model
⇒ Search & Definition of Key Concepts
Example 1: Flight Monitoring Domain Model

Aircraft → Route → Fix

3D-Point
Example 1: Flight Monitoring Domain Model
Example 1: Flight Monitoring Domain Model

Aircraft → Flight Plan → Route → Fix → 3D-Point → Real-Time Tracking → Collision Avoidance

High-Level Domain Model
Example 2: Domain Model for a Financial Institution
Problem:
Model-Explosion. ⇒ Size of the models grows!

Build hierarchical models
Enterprise Level

Domain Level

PartnerContext

Segmentation

PartnerGroup

Instruction

Address

AdressingInstruction

Servicing

Contact

Refinement

Partner

Dossier

PartnerAgreement

Party

Agreement

VariousData

Compliance

Hierarchy:

2nd Level
Building a successful Domain Model

Fair, constructive and open dialog

Identify and describe:
• Business concepts in the domain
• Relationships
• Attributes and contraints
Is behaviour part of a Domain Model?

Yes!
The variables, constraints and operations on domain concepts must be identified and specified
Is behaviour part of a Domain Model?

⇒ Model enrichment
Model ⇒ Code

Domain Model

variables
operations
constraints

mortgage
account

customer
owns

Checking account

variables
operations
constraints

«Having created a great model, but failing to properly transfer it into code will end up in software of questionable quality»

```java
try {
    Object homeId = lookup("ejb:session/creditRating");
    Class cls = Class.forName(
        "java.dbc pstmt.restrictions.CreditRatingService");
    CreditRatingService ratingService = (CreditRatingService)
        PortableRemoteObject.narrow(homeId, cls);
    if (ratingService == null)
        addAuditTrailEntry("Failed to lookup 'ejb:session/creditRating'" + ".
            Ensure that the bean has been " + "successfully deployed");
    return;
}
```

```
CreditRatingService ratingService = ratingHome.create();
```

// Retrieve som from store
Element elem =
    (Element) getElement("input", "payload", "som");

int rating = ratingService.getRating();
```
```
```
Continuous Evolution of the Domain Model

Richness in Content

Richness in Expression

$M_1$ $t_1$

$M_2$ $t_2$

$M_n$ $t_n$
DSE: Loss of Consistency

The code must be an expression of the model

A change in the code may need a change in the model
Anticorruption Layer =

Anti-Corruption Layers maintain the **integrity** of differing systems and models.

**Semantic Mismatch**

**Concept inconsistency** etc.
Anticorruption Layer

Explicit Mapping between contexts and code

Domain or System «A»

Concepts & Models

Domain or System «B»

Concepts & Models
Example:
Concept «Time»

System «A»
- Reference: DCF77
- Resolution: 1 sec
- Operation: continuous

DCF

System «B»
- Reference: quartz impulses
- Resolution: 1 msec
- Operation: counter
- Nulled at power-up

E287A19B5

Example:
Concept «Time»

Anticorruption Layer

Get{time}
Example:
Concept «Time»

Agree on:
- Common (external) reference
- Exchange format

System «A»
- Reference: DCF77
- Resolution: 1 sec
- Operation: continuous

System «B»
- Reference: quartz impulses
- Resolution: 1 msec
- Operation: counter
- Nulled at power-up

Reference: E287A19B5
**Summary:** The DSE concepts:

- Business/Application Domain
- Bounded Context
- Domain Model
- Anticorruption Layer

---

- **Sphere of Knowledge, Influence or Activity**
- **The Bounded Context is the explicit Boundary of a Model**
- **“Formal” representation of the Entities, Relationships and their Properties in your Domain**
- **Maintains the integrity of differing systems and models**
DSE

Fundamentals (2/2)
The Tools:

Context Map

Ubiquitous Language

Domain-Specific Language

DSE Patterns
Context Map =

A Context Map is a document which defines and delineates the different bounded contexts for systems/models and the relationships between them.
Example: Context = Country

Bounded Context = Country

Concepts =
- Language
- Laws
- …

“Pool” =
Schwimmbecken
Fonds
Poolbillard
Bassin
Tümpel
Einsatz
Fundus
gemeinsame Spielkasse
Grube
Interessengemeinschaft
Konsortium
Reservoir
Ring

http://dict.leo.org
Can we model the whole world in one model?

A complete Enterprise?

A department?

**NO** – We need clearly defined *boundaries* for our models
Context Map

Context Map = Definition of different bounded contexts and the relationships between them

Bounded Context A
Application Area 1: MODEL A

Bounded Context B
Application Area 2: MODEL B

Bounded Context C
Application Area 2: MODEL C
**Example:** Internet-Sales

1. **Domain Model**
   - Shipping Subdomain: Packaging
   - Delivery Subdomain
   - Pricing Subdomain: Discounts
   - Sales Subdomain: Purchasing, Bundles, Claiming

2. **Context Boundary**

3. **Model in Context**
   - Inventory
   - Legacy Context
   - ERP

4. **Legacy Context**
A major reason for failure of software projects is a failure of people = the failure to communicate

**Ubiquitious Language**

The Ubiquitous Language is a *shared* language between the business and the development teams

The Ubiquitous Language comes from the *business*, and is enriched by the development team
Ubiquitous Language

- Business vocabulary
- Implicit knowledge
- No model

- IT vocabulary
- Implicit knowledge
- IT models

Serious communications gap

Business Customer
Needs, Requirements

Information Systems Engineers
Specifications, Implementation

http://mayrsom.com
Future-Proof Software-Systems: Domain Software Engineering

Customer/Business

IT Organization

Domain Experts

Ubiquitous Language

Software Teams

形成化

low

high

«Boxes & Lines» Text

Boxes & Lines with semantics

UML, SysML

Ontologies

Prof. Frank J. Furrer - WS 2015/16
How is an Ubiquitous Language developed?

... very often a good **start** is a textual table

### High Level Domain Entities (Enterprise Level)

<table>
<thead>
<tr>
<th>Domain Concept</th>
<th>Description</th>
<th>Operations</th>
</tr>
</thead>
</table>
| **Organization Entity** | Legal Entity for executing business | • Create the entity  
• Internal organization of the entity  
• Agreements with other parties  
• Creation of financial products  
• Collaborate with other parties  
• Create reports  
• ... |

**Concepts**

**Definition**

<table>
<thead>
<tr>
<th>Domain Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>Value-transferring activity with adherence to legal &amp; regulatory requirements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operations</th>
</tr>
</thead>
</table>
| • Define parties  
• Oblige parties  
• Check legal & regulatory requirements  
• Execute operation  
• Document & archive operation  
• ... |

etc.

etc.
Future-Proof Software-Systems: Domain Software Engineering

Domain Experts
Ubiquitous Language
Software Teams

Formalization

Textual Table

«Boxes & Lines» Text
Boxes & Lines with semantics
UML, SysML
Ontologies DSL’s

«Boxes & Lines»

Text
How is an Ubiquitous Language developed?

**Content**: Concepts, Behaviour, Constraints

**Form**: Syntax, Enrichment, Precision

- Domain Expert
- IT Expert
Domain Specific Language =

A computer programming language of limited expressiveness focused on a particular domain.

The domain focus is what makes a limited language worthwhile.

Fowler/Parsons 2011
Domain-specific languages (DSLs) are currently being developed for many application domains, e.g. insurance, banking, robotics, ...

⇒ They *may* prove useful in specific fields of application (Risk: Dilution)
DSE Patterns =

Tried and true design and development reference solutions for Domain-Driven Design (& Domain Software Engineering)
Eric Evans DDD Pattern Diagram:

Eric Evans:
[last accessed: 29.1.2016]
DSE

Alignment & Continuous Integration
Loss of Consistency: Continuous Alignment

Time, Evolution

Business Model ⇔ Domain Model ⇔ Code
must be in sync at all times
Continuous Alignment

Software Development Process

Model Integrity:
«It is so easy to start from a good model and progress towards an inconsistent one»
... and the correctness of the software becomes suspect and its agility is damaged
Continuous Integration

«It is easy to make mistakes when we do not focus 100% on the purity, integrity and consistency of the model»

**Continuous integration** is based on integration of concepts in the model, then finding its way into the implementation where it is tested.
Consequences for Industrial Software Development
Domain Software Engineering (DSE) is an architectural methodology for evolving a software system that closely aligns to business domains.

- Massively business-oriented
- Strongly model-based/model-driven
- Continuous integration
DSE Expected Benefits:

✓ Precise requirements
✓ Minimal divergence business reqs ⇔ IT implementation
✓ Reduction of accidental complexity
✓ Significant increase in software quality
✓ Optimum agility/maintainability of the software
✓ Excellent understandability
DSE Obstacles:

- Strict discipline needed
- Willing business partners
- Adequate development process & governance
- Very competent people
- Continuous refactoring
- Up-front investments (for each project)
Is DSE difficult?

**YES**

Is DSE worthwhile?

**YES** – but needs a strong commitment and much company discipline
DSE

Conclusions
What have we learned?

- The Software Domain Engineering Concepts
- The Software Domain Engineering Tools
- The Pro’s and Con’s of DSE

Why is this knowledge good for you?

- DSE is becoming an important methodology in the (near?) future
- DSE has a number of wonderful concepts
- Introduction of DSE needs time/effort
DSE

References
### References (1/3):

<table>
<thead>
<tr>
<th>References</th>
<th>Authors/Books</th>
</tr>
</thead>
</table>
## References (2/3):

<table>
<thead>
<tr>
<th>References</th>
<th>Authors</th>
<th>Book Title</th>
<th>Publisher, Location, USA</th>
<th>ISBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reinhartz13</td>
<td>Iris Reinhartz-Berger, Arnon Sturm, Tony Clark, Sholom Cohen, Jorn Bettin (Editors)</td>
<td><em>Domain Engineering – Product Lines, Languages, and Models</em></td>
<td>Springer-Verlag, Berlin</td>
<td>978-3-642-36653-6</td>
</tr>
<tr>
<td>Scotts14</td>
<td>Jason Scotts</td>
<td><em>Domain-Driven Desing – How to easily implement Domain-Driven Design</em></td>
<td>Printed in Germany by Amazon Distribution, Leipzig</td>
<td>978-1-631-87691-2</td>
</tr>
<tr>
<td>Avram06</td>
<td>Abel Avram, Floyd Marinescu</td>
<td><em>Domain-Driven Design - Quickly</em></td>
<td>C4Media Inc., USA</td>
<td>978-1-4116-0925-9</td>
</tr>
<tr>
<td>Gonzales08</td>
<td>Cesar Gonzales-Perez, Brian Henderson-Sellers</td>
<td><em>Metamodelling for Software Engineering</em></td>
<td>John Wiley &amp; Sons, Chichester, UK</td>
<td>978-0-470-03036-3</td>
</tr>
</tbody>
</table>
## References (3/3):

<table>
<thead>
<tr>
<th>Reference</th>
<th>Source</th>
</tr>
</thead>
</table>
| Fowler11    | Martin Fowler: *Domain-Specific Languages*  
| OSE03       | Operating System Engineering: *Domain Engineering*  
2003, Downloadable from: https://www4.cs.fau.de/Lehre/SS03/V_OSE/Skript/03ose-A5.pdf [last accessed: 2.12.1015] |
| Henderson12 | Brian Henderson-Sellers: *On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages*  
| Rumbaugh05  | James Rumbaugh, Ivar Jacobson, Grady Booch: *The Unified Modeling Language UML Reference Manual*  
| Sølvberg10  | Arne Sølvberg: *Domain Engineering: What is it?*  
| Bjørner05   | Dines Bjørner: *Domain Engineering*  
| Taylor10    | Richard N. Taylor, Nenad Medvidovic, Eric M. Dashofy: *Intro to Domain-Specific Software Engineering*  
Downloadable from: http://sunset.usc.edu/classes/cs578_2009b/23_Intro_to_DSSE.ppt |
Questions please