Designing complex softwares.

PLAN

[Abstract: 2](#_Toc15389191)

[Introduction: 2](#_Toc15389192)

[I. Research problem 3](#_Toc15389193)

[A) Problem description 3](#_Toc15389194)

[B) What is Design, develop, complex software? 3](#_Toc15389195)

[C) Fined grained view and sub categories of the problem 4](#_Toc15389196)

[II. State of the art 5](#_Toc15389197)

[A) Model Driven Architecture (MDA) 5](#_Toc15389198)

[i) History 5](#_Toc15389199)

[ii) What is MDA? 5](#_Toc15389200)

[iii) Example of application 8](#_Toc15389201)

[iv) Drawbacks of MDA 8](#_Toc15389202)

[B) Behavior Driven Design (BDD) 8](#_Toc15389203)

[i) History 8](#_Toc15389204)

[ii) What is BDD? 8](#_Toc15389205)

[iii) Example of application 8](#_Toc15389206)

[iv) Drawbacks of BDD 8](#_Toc15389207)

[C) Domain Driven Design (DDD) 9](#_Toc15389208)

[i) History 9](#_Toc15389209)

[ii) What is DDD? 9](#_Toc15389210)

[iii) Example of application 9](#_Toc15389211)

[iv) Drawbacks of DDD 9](#_Toc15389212)

[III. Research Methodology 9](#_Toc15389213)

[A) Qualitative analysis 9](#_Toc15389214)

[B) Quantitative analysis 9](#_Toc15389215)

[C) Results and comments 9](#_Toc15389216)

[IV. Assessment and validation of results: case of AXA Bank DSI 9](#_Toc15389217)

[V. Conclusion and further work 9](#_Toc15389218)

[Table of acronyms 9](#_Toc15389219)

[Bibliography: 10](#_Toc15389220)

[Appendices 10](#_Toc15389221)

[A) Semi-structures Interview Guide 10](#_Toc15389222)

[B) Questionnaire for the quantitative analysis 10](#_Toc15389223)

# Abstract:

# Introduction:

According to the IT context, a complex software is considered to be related to critical business areas (health, insurance, banking), where business knowledge is held by a small group of industry experts. In addition, the software is designed with a distributed architecture, allowing the deployment and execution of the latter on several platforms and on different environments. Such software is also able to handle traffic of millions of users and run 24/7.

We use complex software platforms on a daily basis (Airbnb, Leboncoin, platforms and mobile applications for banking services) and we cannot ignore how vital they are to us. Complex software is undeniably important, and so are the challenges to developing it. Addressing the issues implied by these challenges is therefore inevitable. Over the past 30 years, we have seen a series of project management methodologies and software design models emerge to solve these issues that are of several types: from project management to software design patterns and technical coding best practices. However, to get to know these practices, you have to read tons of documents, sometimes not clear enough and not very explicit, which can lead to misunderstandings, abandonment of the learning process, misuse and finally facing the same issues that the methodologies where supposed to solve.  
This paper did focus on software designs patterns and technical development best practices with the aim to produce guidelines with concrete step by step methodology.  
Methodologies have been elicited and categorized by the kind of issues they are addressing:  
Design issues, Software implementation related issues, and Methodology concreteness related issues, then thanks to qualitative and quantitative research through semi-structured interviews and questionnaires, feedbacks from stakeholders of complex software projects have been extracted with the most important methodologies and practices in order to provide the guidelines, that have been assessed through a bank loan project at AXA Bank.

# Research problem

## Problem description

Companies seek building customer’s loyalty, supply ever better products and services on an ongoing basis. Hence they face the challenge of creating software for critical domains such as health, finance, e-commerce, loans and mobile banking.

Yet, they are facing poor collaboration between domain experts and development teams, since such collaborations happen only on early project steps. Moreover there is no specific software architecture; the *big ball of mud* pattern is then applied.

As i noticed at AXA bank Information System Department, where I’ve been participating to the designing of Java based applicative systems providing customer bank services, the lack of frequent synchronization between domain expert and technical teams leads to the following problems:

Sluggish pace of feature enhancement, buggy releases and frequent production incidents and Developers firefighting continuously.

As the software community tried to come across these problems, we witnessed the venue of a bunch of designing principles that must be applied to a proper context to prove their usefulness, unfortunately, this decision is not easy for developers, since deciding of a designing principle with respect to the context requires reading and mastering the concepts across years of experiences and tons of documentation.

As a result, developers find themselves applying design patterns either to an improper context:

* Applying advanced architecture pattern to simple problem domains e.g. presentational website, small size school management system, event registration system
* Trying to apply Unknown domain boundaries//look in the book “tactical and strategic patterns…” when and when not to apply
* …

In a nutshell, how can teams Design and Develop complex Software, following concrete steps, in order to tackle the previous problems?

## What is Design, develop, complex software?

Designing is deciding what to build And How to build.

Developing is implementing what have been decided.

A complex Software is characterized with the following:   
- Related to critical business areas (health, insurance, banking),  
- Business knowledge held by a small group of domain experts.   
- Distributed architecture, allowing the deployment and execution of the software on several platforms and on different environments,  
-Handle traffic of millions of users and run 24/7.

According to these definitions, the problem can be reformulated as following:

How to decide what to build and how to build to provide critical services to thousands of users?

How to build and correct quickly what have been decided?

Following clearly defined and concrete steps?

Still, these statements represent the problem in a coarse grained view that we need to zoom in to get more insights about resolution axis.

## Fined grained view and sub categories of the problem

…

How to emphasize collaboration between domain experts and development teams and fill the gap between both teams?

…

How to help the development teams to focus on domain issues as described by domain experts?

How to separate technical implementation concerns from domain logic issues?

How to apply an architectural style facilitating scalability and features enhancements?

…

How to define concrete steps easy to apply?

With fine grained statements comes out that we are facing three kinds of issues:

* Designing issues
* Technical implementation issues
* Approaches concreteness issues

Getting to know the diverse solutions suggested before now requires analyzing what have been done so far in companies and projects.

# State of the art

…

## Model Driven Architecture (MDA)

### History

…

The MDA initiative has been created because, as it was existing many Meta models languages being developed independently such as UML, OCL, the fear of inconsistency and compatibility between different Meta models was growing. The need of synchronizing this was to develop a metamodel definition language: a meta-metamodel.

…

### What is MDA?

#### Definition

…

#### Methodology

…

Model Driven architecture is an approach to using models in software development to produce applications independent of the infrastructure they use. [1]

A Meta model defines the language with what the model will be built.

In software engineering, UML is a metamodel defining the language used for describing Object oriented software artifacts.

Before MDA standards, models were used in a contemplative way, meaning models were designed, and address to the developers as an inspirational tool. This is still often done today.

Ex: Designing class Diagrams, showing the developers what classes are to develop.

MDA core standards as XMI((XML metadata interchange), JMI ,Unified Modeling language(UML),Meta object facility (MOF), Common Warehouse metamodel(CWM)... help developments being more aligned on models.

Basic principles/goals of MDA:

Everything is a model.

A model can be transformed to another model based on Meta models.

Meta models help separation of concerns.

Separation of concerns between business concepts and platform (technical) aspects each corresponding to a metamodel.

Separation of implementation details from business functions.

Make applications independent of the infrastructure they use.

Description of the developed system by specific domain/business language concepts.

Convert automatically business based models to platform/technical specific models.

Ex: From UML class model to JAVA Code using JMI standard or TO XML using XMI.

Not focusing on programming languages only but modeling languages too.

Programming languages, by themselves cannot handle separation of concerns, except at a purely syntactic level.

Models are not perceived as simple documents anymore as they are reusable and portable.

According to [2]

Easily integrate new implementation infrastructures into existing designs,

Generate significant portions of application-specific code, configuration files,

data integration bridges and other implementation infrastructure artifacts from models, More easily synchronize the evolution of models and their implementations as the software evolves, and rigorously simulate and test models.

MDA approach as defined in [1]:

Specifying a system independently of the platform that supports it

UML models about business functions: how does the system delivers value

Create Platform Independent Models (PIM)

Specifying platforms and Choosing a particular platform for the system.

Transform UML models to platform specific models by adding specific APIs, Commercial-off-the-shelf (COTS) products.

Transforming the system specification into a particular platform.

Generate code targeting a specific platform from the PIM

Some flaws/pitfalls:

The logic implemented in code is not always represented in models, therefore how to create/maintain the code base while keeping alignment with the models?

Main importance and significance of the

MDA is the portability and interoperability of

The same design on different platforms. MDA enables reuse at the domain level.

MDA building blocks:

- Models driven: uses models to emphasize understanding, implementation and maintenance

- Platform: Set of subsystems and technologies providing a set of functionalities to any application supporting it, whatever its implementation.

- Platform independence

- Viewpoint:

Computation independent viewpoint: supported by a Computation Independent Model (CIM) or Domain model, focusing on the requirements for the system, the details of structure and processing are hidden, Bridging the gap between domain experts and artifacts experts.

Platform independent viewpoint: Supported by the Platform Independent Model (PIM) providing a view of the system from a platform independent viewpoint.

Platform specific viewpoint: Supported by a Platform Specific Model(PSM) : Combining specs from CIM and PIM, defining how the system is using a specific platform.

- Platform model: Specificity about the platform and how to connect the application to the platform.

- Model transformation: Converting one model to another model of the same system. Ex: PIM=>PSM

MDA models can be expressed using the Unified Modeling Language (UML).

MDA main drawback: Software customization Issue: Inconsistency between the PIM and the artifacts produces through customization.

…

### Example of application

…

### Drawbacks of MDA

## Behavior Driven Design (BDD)

### History

### What is BDD?

#### Definition

#### Methodology

### Example of application

### Drawbacks of BDD

## Domain Driven Design (DDD)

### History

### What is DDD?

#### Definition

#### Methodology

### Example of application

### Drawbacks of DDD

## 

# Research Methodology

## Qualitative analysis

## Quantitative analysis

## Results and comments

# Assessment and validation of results: case of AXA Bank DSI

# Conclusion and further work

# Table of acronyms

BDD: Behavior Driven Design

DDD: Domain Driven Design

ISD: Information System Department

MDA: Model Driven Architecture

TDD: Test Driven Design

UML: Unified Modeling Language

XMI: XML metadata interchange

# Bibliography:

[1] Model driven architecture: Dr. Shahid Nazir Bhatti, Asif Muhammad Malik, April 2015.  
 Available at: <https://www.researchgate.net/publication/274916541_Model_Driven_Architecture>   
[Accessed April 29,2019]

[2]Model-driven Development of Complex Software: A Research Roadmap, Robert France and  Bernhard Rumpe, June 2007  
 Available at: <https://www.researchgate.net/publication/4250888_Model-driven_Development_of_Complex_Software_A_Research_Roadmap>,   
[Accessed April 29,2019]

# Appendices

## Semi-structures Interview Guide

## Questionnaire for the quantitative analysis