Université Paris 1 Panthéon - Sorbonne

MANAGEMENT of INFORMATION AND KNOWLEDGE SYSTEMS

MIKS Master Thesis

Promotion 2018-2019

Enhancing the designing and the development of large-scale softwares.

Presented and defended by: Philippe Fabrice GUEMKAM SIMO

Academic Supervisor: Manuele Kirsch Pinheiro

Defended on: 16/10/2019

ACKNOWLEDGEMENT

I would like to express my gratitude to my academic supervisor Mrs. Manuele Kirsch Pinheiro for the guidance all along this thesis process.

Thanks to my company tutor, Mr. Pierre Nivon, for being a great support whenever it was needed.

Special thanks to the teachers of this master who did their best to make us learn so much in one year.

Another special thanks to Mrs. Selmin Nurcan who is doing her best to allow us to get the most out of this master.

Finally, I thank all of my classmates for having accepted me like a brother in a family that we are.

ABSTRACT

This thesis is discussing about software design methodologies applied to large-scale softwares for helping organization enhance how they design and develop such softwares. For that purpose, diverse software development methodologies have been analyzed in order to focus on the issues that are not already covered by those methodologies. Then a qualitative research powered by semi-structured interviews has been proceeded. The interviews were recorded with persons from diverse backgrounds but having actively been involved in the development of large-scale software. Finally, driven by the insights from the state of the art and a horizontal analysis on the records, guidelines have been suggested and assessed in the specific context of the information system department (ISD) of AXA Bank.

PLAN

ACKNOWLEDGEMENT i
ABSTRACT ii
PLAN iii
List of illustrations v
List of tables vi
INTRODUCTION 7
1. Research problem 9
1.1) Problem description 9
1.2) What is Design and develop large-scale softwares? 9
1.3) Fined grained view and sub categories of the problem 9
2. State of the art 11
2.1) Model Driven Architecture (MDA) 11
2.1.1) History 11
2.1.2) What is MDA? 12
2.1.3) Drawbacks of MDA 15
2.2) Behavior Driven Design (BDD) 15
2.2.1) History 15
2.2.2) What is BDD? 15
2.2.3) Drawbacks of BDD 19
2.3) Domain Driven Design (DDD) 20
2.3.1) History 20
2.3.2) What is DDD? 21
2.3.3) Drawbacks of DDD 26
2.4) Conclusion 27
3. Research Methodology: Qualitative analysis 29
3.1) Methodology 29
3.1.1) The questionnaire and the interview 29
3.1.2) Organization of the result grid (themes and sub-themes) 30
3.2) Analysis 30
3.2.1) Stakeholder's roles vs. activities: 30
3.2.2) Knowledge on approaches 31

3.2.3) Usage of software designing approaches 31
3.2.4) Approaches satisfaction feedback: 31
3.2.5) Issues and expectations: 31
4. Discussion 32
4.1) Key Takeaways from the analysis 32
4.2) Guideline proposal 35
4.2.1) Preparing the organization 35
4.2.2) Setting up the Knowledge Management (KM) System 36
4.2.3) Designing and developing 37
5. Assessment of guidelines: case of AXA Bank ISD 43
5.1) Organization refactoring within AXA Bank ISD 43
5.1.1) Overview 43
5.1.2) Team and Technical Agility transformation 44
5.2) Knowledge Management (KM) 46
5.3) Trainings around event storming and applying the hexagonal architecture 46
5.3.1) Training Workshops 46
5.3.2) Keep the teams up to date with Coding Sessions 47
5.4) Key Performance Indicators (KPI) and appraisal 47
6. Conclusion and further work 49
List of acronyms 50
Bibliography 51
Appendices 53
Semi-structured Interview Guide 53
Qualitative analysis summary 55

List of illustrations

Figure 1: Project failure rate from chaos report 2015 7
Figure 2: The more complex and bigger the higher the risk of failure. 8
Figure 3: Model Driven Architecture 14
Figure 4: MDA Methodology 14
Figure 5: A feature “Is it Friday yet” with steps defined in Gherkins 16
Figure 6: How Cucumber Works 17
Figure 7: “Is it Friday yet” corresponding steps definition in a java test file 18
Figure 8: Behavior Driven Development methodology 19
Figure 9 Monolithic Architecture 20
Figure 10: Two bounded Contexts independent from each other 22
Figure 11 : One business bounded context, one micro-service, one domain model, one database 23
Figure 12: Overview of Hexagonal architecture. 24
Figure 13: Extended Hexagonal Architecture: The Onion Architecture 25
Figure 14: Micro-services searches trend since 2004 26
Figure 15: What interviewed where likely to talk about 30
Figure 16: From pyramidal architecture to decentralized architecture 36
Figure 17 Stakeholders at the same organization level 36
Figure 18: Event storming legend 38
Figure 19: Micro-service coupled to onion architecture 39
Figure 20: Team and technical Agility Process of AXA Bank ISD 44
Figure 21: The result of an event storming 47

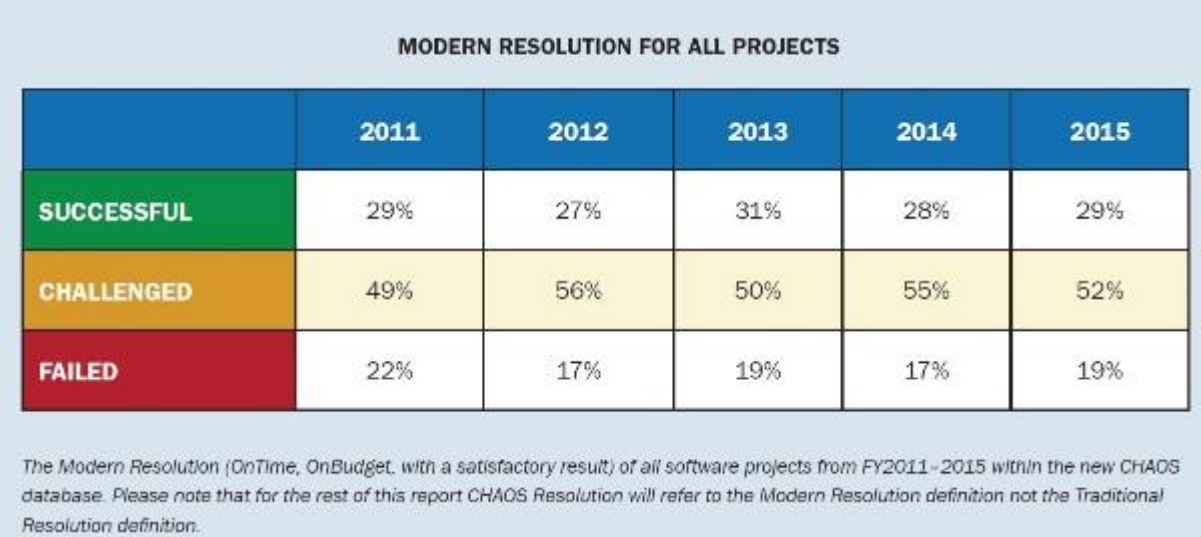
List of tables

Table 1: The research problem distilled.....10
Table 2: Existing approaches with respect to the research problem.....28
Table 3: Research Summary.....33
Table 4: Overview of suggested guidelines.....41
Table 5: Knowledge management within the ISD.....46
Table 6: Appraisal of some guidelines within AXA Bank ISD.....48

INTRODUCTION

According to the IT context, a large-scale 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 over several platforms and diverse environments. Such software can handle traffic of thousands of millions of users and run 24/7.

We use large-scale software platforms on daily basis (Airbnb, Leboncoin, platforms and mobile applications for banking services) and we cannot ignore how vital they are to us. Large-scale softwares are undeniably important and so are the challenges for developing them. 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 practices trying to solve these issues that are of several kinds. However, we are still facing some project failures due to requirements not met, products not easily maintainable, misuse of technologies as shown in the Figure 1 and Figure 2 from the 2015 chaos report.



MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

Figure 1: Project failure rate from chaos report 2015[1]

[1] Source : https://res.infoq.com/articles/standish-chaos-2015/en/resources/Modern%20Resolution.jpg

		COMPLEXITY				
		C1	C2	C3	C4	C5
SIZE	S1	100	250	400	550	700
	S2	175	325	475	625	775
	S3	250	400	550	700	850
	S4	325	475	625	775	625
	S5	400	550	700	850	1000

Figure 2: The more complex and bigger the higher the risk of failure.[2]

This paper did focus on software development methodologies and software designs patterns with the aim to suggest a set of guidelines in order to enhance the process of developing large-scale software. For that purpose, methodologies have been elicited and categorized by the kind of issues they are addressing - Design issues and Software implementation related issues - then thanks to a qualitative research through semi-structured interviews, feedbacks have been gathered and insights have been discovered, leading us to suggest some guidelines. Finally, the guidelines have been assessed with respect to the practices experienced at AXA Bank.

[2] Source : https://res.infoq.com/articles/standish-chaos-2015/en/resources/Complexity%20Matrix.jpg

1. Research problem

1.1) 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 softwares for critical domains such as health, finance, e-commerce, loans and mobile banking. Yet, they are constantly facing customer complaints about products, frequent incidents and frequent project abandon in favor of new ones.

The same kinds of issues were also encountered within the AXA bank ISD, where, I've been participating to the designing of many Java based systems providing bank services to customers. More specifically, the ISD was facing, sluggish pace of feature enhancement, buggy releases, frequent production incidents and developers firefighting continuously.

The issue to tackle in this paper is about how can companies improve their way of designing and developing large-scale softwares?

1.2) What is Design and develop large-scale softwares?

Designing is deciding what to build and how to build. Developing is implementing what have been decided. A large-scale software is usually related to critical business areas (health, insurance, banking), with the business knowledge held by a small group of domain experts; moreover it is built over a distributed architecture, allowing its deployment and execution on several platforms and diverse environments; finally, it can handle traffic of thousands of millions of users and run 24/7.

According to these definitions, the problem can be reformulated as following: How to enhance the process of deciding what to build, how to build and maintaining a platform that will provide critical services to millions of users?

Still, these statements represent the problem in a coarse-grained view that we need to zoom in.

1.3) Fined grained view and sub categories of the problem

Getting to know a more fine-grained view of the problems will definitely help us answering the good questions.

Deciding what to build and how to do it relies heavily on the collaboration quality between domain experts and the development team in order to fill the gap between the both. Usually, the problem is about how to educate the teams and let them notice the importance and the priority of getting aligned with the business? How to speak to the business people and stakeholders in order to captivate their interest and get the most correct domain knowledge as possible?

When it comes to implement a maintainable solution, according to the requirements, the softwares development patterns come into play. The relevant questions are: which development pattern helps the team 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?

With fine-grained statements comes out that we are facing two general kinds of issue, which are the designing issues and the technical implementation and maintenance issues. The table 1 summarizes the research problem.

Table 1: The research problem distilled

Research problem	**Coarse Grained Issues**	**Fine Grained Issues**
How can companies improve their way of designing and Developing large-scale Software?	**How to decide what to build and how to build to provide critical services to thousands of users?** **Designing issue**	**How to educate the teams and let them notice the importance and the priority of getting aligned with the business?**
		How to speak to the business people and to captivate their interest?
		How to get the most correct domain knowledge as possible?
	How to build and correct quickly what have been decided? **Implementation and maintenance issue**	**Which development pattern helps the team 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?

Getting to suggest solutions to the issues requires first analyzing what have been done so far in companies and projects.

2. State of the art

As the software community tried to come across the research problems, we witnessed the venue of a bunch of designing principles that must be applied to a proper context to prove their usefulness.

2.1) Model Driven Architecture (MDA)

2.1.1) History

A model is a set of simplified representations of a reality that is too complex to understand as is. A model is supposed to be understood by everybody, including people not holding enough knowledge about the related domain. A Meta model defines the informations such as the language needed to build, read and understand the model. In software engineering, the Unified Modeling language (UML) is a meta-model defining the language used for describing object oriented software artifacts.

Before MDA, models were used in a contemplative way, meaning models were designed and addressed to the developers as an inspirational tool (Bhatti & Malik, 2015). A common illustration is about designing UML class Diagrams at the beginning of the project and showing the developers what classes are to develop. Beside the need of making models more useful than a simple inspirational tool, many meta-models languages were developed independently such as UML and the Object Constraint Language (OCL) causing the fear of inconsistency and compatibility between different meta-models growing as well as the need of synchronization. (Bhatti & Malik, 2015)

2.1.2) What is MDA?

2.1.2.1) Definition

The Object Management Group™ (OMG™) was formed as a standards organization to help reduce complexity, lower costs and hasten the introduction of new software applications (Truyen, 2006).
Model Driven architecture is an approach introduced by the OMG, to using models in software development to produce applications independent of the infrastructure they use. (Bhatti & Malik, 2015)

MDA specifies three default models (Truyen, 2006):
Computation Independent Model (CIM) A CIM uses a vocabulary that is familiar to the domain terms. It presents what the system is expected to do but hides all information technology related specifications to remain independent of how that system will be implemented. The CIM plays an important role in bridging the gap which typically exists between these domain experts and the development team responsible for implementing the system.

Platform Independent Model (PIM): A PIM exhibits enough degree of independence so as to enable its mapping to one or more platforms. This is commonly achieved by defining a set of services in a way that abstracts out technical details. Other models then specify a realization of these services in a platform specific manner.

Platform Specific Model (PSM) A PSM stipulates how a system uses a specific platform.

The aim of MDA is to facilitate the creation of machine-readable models with a goal of long-term flexibility. MDA allows the use of one set of models that are independent of technical infrastructure to specify the features of an application and then generate many applications according to the targeted technical infrastructure. Giving this way, supplementary power to the models. Besides, the models remain stable as technology evolves, extending and thereby maximizing software portability, interoperability and return on investment (Object Management Group, 2001).
As models rely on meta-model, MDA do too. The latter supports four core standards/meta-models (Figure 3) namely XML metadata interchange (XMI), Unified Modeling language (UML), Meta Object Facility (MOF) and Common Warehouse meta-model (CWM) which help developments being more aligned on models. (Object Management Group, 2001)

MDA has core principles giving it its usefulness: Everything is a model, a model can be transformed to another model, there must be a separation of concerns between business concepts and platform (technical) aspects, a business-based model can be converted to platform/technical specific models by generating significant portions of application-specific code and configuration files. (France & Rumpe, 2007)
An example can be to generate JAVA class model or XML model from UML class diagram.

2.1.2.2) Methodology

Defining the platform independent models:

Software development in the MDA starts with a Platform-Independent Model (PIM) of an application's business functionality and behavior. (Object Management Group, 2001) That is specifying a system independently of the platform that supports it, then creating the models (using MDA's standard modeling specifications) that highlights how does the system deliver value: the business functions to be implemented. These models will survive to platforms related changes as they are entirely independent; as well as UML models (Use case diagrams, class diagrams) will survive if a change from JAVA to C++ programming language occurs for coding the system. This first step occurs in the center disk of the MDA overview from Figure 3.

Defining transformation mappings:

Once the PIM have been specified, they are converted to a Platform-Specific Model (PSM) and then to a working implementation on virtually any middleware platform: Web Services, XML/SOAP, EJB, C#/.Net, OMG's own CORBA®, or others. (Object Management Group, 2001)**.** For that purpose, mappings provide specifications on how to do the transformation. The mappings define rules, called annotations according to MDA parlance that are used for guiding the transformation to a PSM.

Example: A UML PIM to an Enterprise Java Bean (EJB) PSM: marking a UML class with the Stereotype "Session" would result in the creation of a session bean — and other supporting classes — within the PSM.

Marking models:

Marks are additional rules that are required in order to produce the desired output in the PSM.

Recording the transformations:

This is about indicating which PIM were mapped to which PSM and include the mapping rules used for each part of the transformation.

Generating the code:

The final step is to generate the implementation code from the PSM that computers can run. This is done by using MDA development tools, inferring platform specific rules, available now from many vendors. The whole process is illustrated in Figure 4.

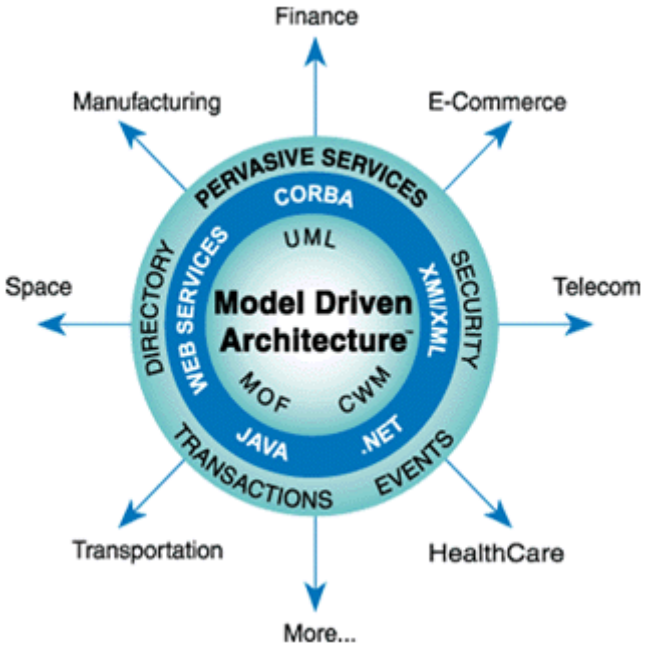


Figure 3: Model Driven Architecture [3]

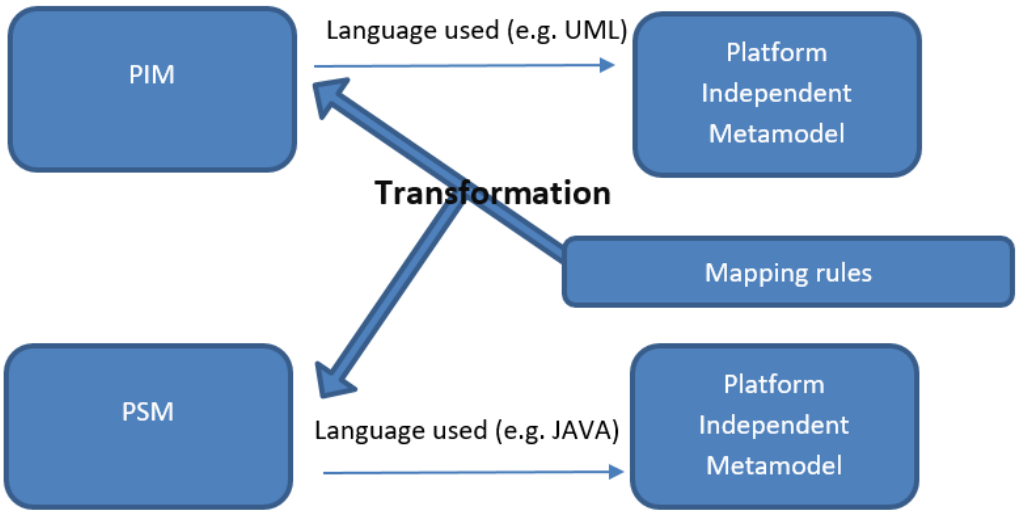


Figure 4: MDA Methodology

How does MDA bring answers to our fined grained problems?

MDA is Models driven: it uses models to emphasize understanding, implementation and maintenance hence bridging the gap between domain experts and artifacts experts. Moreover, it helps focus on the requirements for the system, the technical implementation details being hidden.

[3] Source : https://www.omg.org/mda/

2.1.3) Drawbacks of MDA

Although MDA has been widely accepted and used in the software development world, it presents a non-exhaustive set of pitfalls.

Customizing the generated PSM lead to an inconsistency between the PIM and the PSM: The domain logic implemented in code is not always represented in models, therefore it raises the problem of to create/maintain the code base while keeping alignment with the models.

MDA relies on a variety of technical standards nevertheless, some of which are incomplete or not yet specified or implemented (Thomas, 2003).

We should also notice that MDA requires specialized skillsets: Practitioners of MDA based software engineering are required to have a high level of expertise in their field, such profiles are scarce compared to the availability of traditional developers (Ambler).

2.2) Behavior Driven Design (BDD)

2.2.1) History

During the 1990s, the term "Specification by Example" started floating around. Specification by example is well emphasized by the following quote: "We use realistic examples as a single source of truth for requirements and automated tests on software projects" – ***Ward Cunningham*, A Pattern Language of Competitive Development *(1996)***

Specification by Example was a game changer in the software development world as it allowed for a more precise method of describing and defining requirements. Testers and developers were able to extract information about requirements through concrete examples rather than abstract specifications. (Wanivenhaus, 2017)

Later, Test-Driven Development (TDD) coined by Kent Beck was introduced. Nevertheless, this approach was completely driven by technology and was more of a programmer's discipline rather than that of a tester's – however, it was also missing a high level and business readable aspect. In 2006, Dan North introduced Behavior-Driven Development (BDD) to fill this void by placing specification on a business level – not through code, but business behavior. (Wanivenhaus, 2017)

Nowadays: BDD is widely used as it allows teams to create living requirements documentation that is easy to maintain and can be consumed by all team members, including testers, developers and product owners.

2.2.2) What is BDD?

2.2.2.1) Definition

BDD is a specification technique that certifies that all functional requirements are implemented properly, through the connection of the textual description to automated tests. (Atem de Carvalho, Luiz de Carvalho e Silva, & Manhaes, 2010)

BDD relies on Test-Driven Development (TDD), which is a technique consisting of defining test cases for functionalities before they get implemented. This way, each implementation increment will get tested against the written cases and must meet the tests requirements to be validated.

2.2.2.2) Methodology

Applying BDD to the design of software starts by identifying the expected behaviors of the system that are more concrete and easier to discover, from the business outcomes that the system intends to produce. The behaviors are then split into a set of features that indicate what should be done to achieve the business outcome. This step relies heavily on discussions between business analyst, business people, testers and developers on business outcomes as illustrated on steps 1 of Figure 8.

As illustrated by the step 2 of Figure 8, once the features have been elicited, they are converted to plain text natural language description as user stories and scenario templates using a specific format: The Gherkins language as illustrated in Figure 5.

A story/scenario defined in Gherkins follows this template:
For a user story:
[Story Title] (One line describing the story)
As a [Role]
I want a [Feature]
So that I can get [Benefit]

For a scenario:
Scenario 1: [Scenario Title]
Given [Context]
And [Some more contexts]....
When [Event]
Then [Outcome]
And [Some more outcomes]....
Scenario2: [Scenario Title]… (North, 2006)

```
Feature: Is it Friday yet?
  Everybody wants to know when it's Friday

  Scenario: Sunday isn't Friday
    Given today is Sunday
    When I ask whether it's Friday yet
    Then I should be told "Nope"
```

Figure 5: A feature "Is it Friday yet" with steps defined in Gherkins

After the previous step, as illustrated in step 3 of Figure 8, the mapping of scenarios into tests that are automatically executed happens. For this purpose, a toolkit - such as Cucumber whose features are described in Figure 6: How Cucumber Works Figure 6 - is used to parse

the descriptions and map them to a programming language description, which in turn is used to define automated tests that a program must meet to be validated.

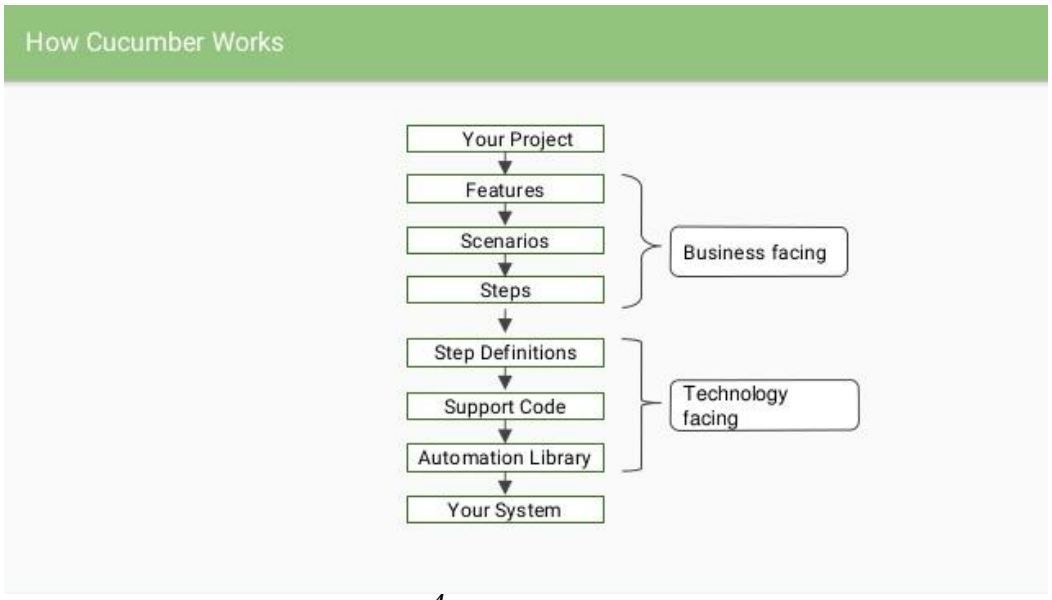


Figure 6: How Cucumber Works [4]

BDD suggests that code should be part of the specifications as well as specifications should be part of the code. (North, 2006) That is, the names of methods and classes in the code should indicate what the method should do, as well as be written in sentences readable by any stakeholder, as illustrated in Figure 7, where is defined the test code for the scenario defined in Figure 5. This is done by sharing the Ubiquitous Language (UL) between code and specifications. The UL is the language which structure comes from the domain model. It helps stakeholders to speak the same language than developers without ambiguity.

[4] Source : https://www.slideshare.net/kalhanrl/cucumber-for-automated-acceptance-testingpptx

```
package hellocucumber;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.When;
import cucumber.api.java.en.Then;
import static org.junit.Assert.*;

class IsItFriday {
    static String isItFriday(String today) {
        return null;
    }
}

public class Stepdefs {
    private String today;
    private String actualAnswer;

    @Given("^today is Sunday$")
    public void today_is_Sunday() {
        today = "Sunday";
    }

    @When("^I ask whether it's Friday yet$")
    public void i_ask_whether_it_s_Friday_yet() {
        actualAnswer = IsItFriday.isItFriday(today);
    }

    @Then("^I should be told \"([^\"]*)\"$")
    public void i_should_be_told(String expectedAnswer) {
        assertEquals(expectedAnswer, actualAnswer);
    }
}
```

Figure 7: "Is it Friday yet" corresponding steps definition in a java test file

Finally, as shown in steps 4 and 5 of Figure 8, the behaviors resulting from the automated tests are validated with the stakeholders, then other items from the backlog are injected and the cycle is ready to start again.

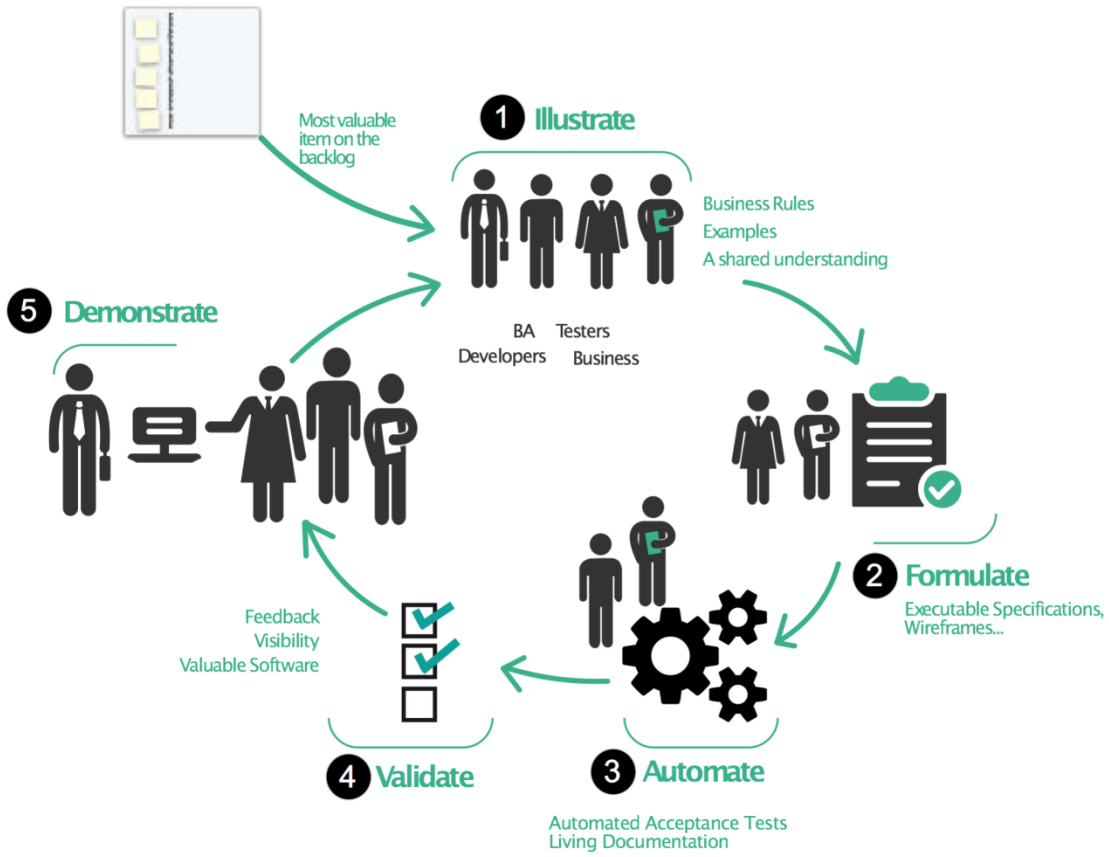


Figure 8: Behavior Driven Development methodology 5

How does BDD bring answers to our fined grained problems?

BDD and TDD are being adopted widely in the software engineering world because they improve software quality and productivity. (Solis & Wang, 2011). By supporting the UL, BDD enhances the collaboration between development teams and stakeholders. Moreover, BDD relies on the process of gathering requirements and checking if they are well implemented, which is done within an iterative process of automated tests checks. Such processes have the effect to drastically ameliorate the quality of the produced software with respect to the cohesion between what is built and what was meant to be built. As any other valuable notions, BDD come with his own costs.

2.2.3) Drawbacks of BDD

BDD has some drawbacks that should not be neglected as they can negatively impact the software development lifecycle.

First it is time consuming: One of the most important steps of BDD methodology is the edition of the feature and scenario files with the Gherkins language. As it requires an overhead investment of time and effort, it may not be worth it for small projects, but for complex projects with lot of iterations, the return on investment of such efforts is quite sure.

[5] Source : https://johnfergusonsmart.com/so-you-say-you-are-doing-bdd-the-story-of-the-whiteboard-and-the-nail-gun/

Secondly, BDD requires intensive communication (Solis & Wang, 2011): There is a need of intensive communication between the person writing the feature files and the person developing the automation code. The coder needs to accurately interpret these files and the scenarios in order to implement them as automation steps. If there is no mutual understanding about the structure and approach being used, problems will arise as the scenarios become increasingly difficult to turn into working automated tests.

Finally, BDD requires high abstraction level skilled people: Although the concept of Gherkins is simple enough to understand since it is just the natural language, still the need of specific skills are required to structure scenarios in a way that facilitates the understanding and the writing of automation code.

2.3) Domain Driven Design (DDD)

2.3.1) History

Many softwares are made following a monolithic data centric architecture as illustrated in Figure 9 (Boissinot, 2019). Such architecture is characterized by a presentation layer serving the end user, powered by a layer handling all the required business functions – named after the service layer – followed down by another layer – the Data Access Object (DAO) Layer – which feeds the service layer with data from the database and saves data to the database as well.

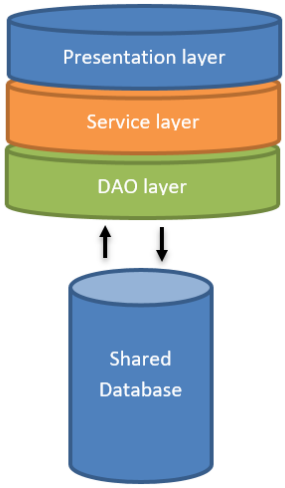


Figure 9 Monolithic Architecture

Such architecture, in large-scale projects, lead, on one hand, to huge and fat components where every implementation is stacked in the same layer, that are painful to maintain and scale up; on another hand, a shared database which decreases the robustness, since, a database in production issue will affect all the system.

As part of a resolution to this issue, The Domain-Driven Design has been introduced by ***Eric Evans in 2003*** in his book ***Domain-Driven Design***: ***Tackling Complexity in the heart of software)***

2.3.2) What is DDD?

2.3.2.1) Definition

The DDD is a philosophy consisting of enabling software developers, to effectively manage the construction and maintenance of software for complex problem domains (Millet & Tune, 2015)**.** It helps achieving not only the challenge of understanding a problem domain but also the creation of a maintainable solution that is useful to solve problems. The power of DDD is its strategic and tactical patterns that can be used to deal with the aforementioned challenges.

2.3.2.2) Methodology

The DDD approach is based on three main pillars that are the Collaborative Modeling, the Strategic Design and the Tactical Design. (Boissinot, 2019)

The first two items are about helping the project team to discover and understand a problem domain, we talk about the problem space**;** meanwhile, the last is about giving the teams the guidelines in order to create a maintainable solution bringing value to the business: The solution space

Collaborative modeling

Domain Driven Design (DDD) is a business designing approach driven by the use cases in order to make the implicit explicit. (Millet & Tune, 2015) The main goal of Collaborative modeling is to elicit use cases. This is done by exploring the existing, discussing with business people to understand them. Some of the used techniques are event storming, agile workshops, impact mapping and user storytelling. Interaction between developers and business guys is highly needed here (Evans, 2004).

The strategic Design

Once the use cases are extracted, they need to be separated within diverse contexts, each being independent from the other, discussed by the UL and treating of one and only one subdomain area (e.g. Sales or Support) of the whole business: They are the Bounded Contexts (BC) as illustrated in Figure 10. Models are then created using a modeling language (e.g. with UML) and the corresponding code model (e.g. with Java) as well, for each bounded context. this where MDA tools patterns could be used. Models are not created equal as the subdomain's complexity differs from one to another. The most appropriate MDA pattern should be found based on the complexity needs of each subdomain. (Millet & Tune, 2015)

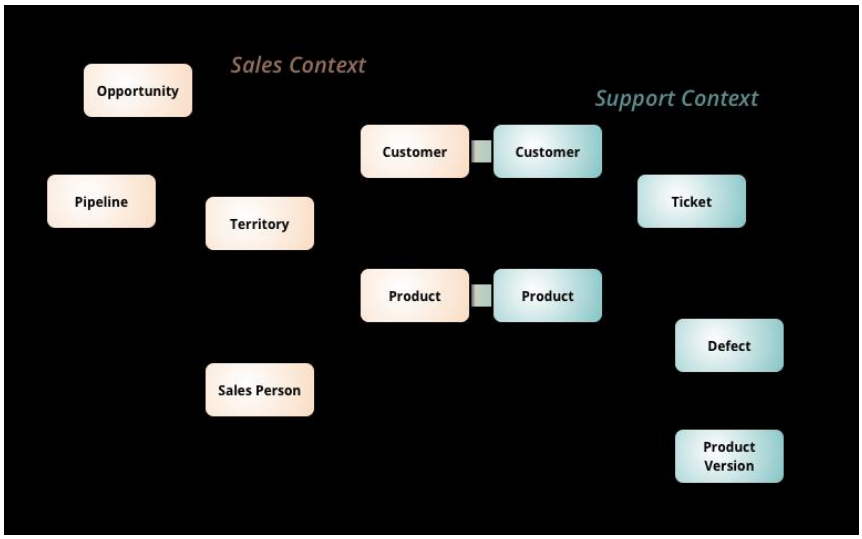


Figure 10: Two bounded Contexts independent from each other[6]

Since models (analysis and code) are built through the collaboration of domain experts and the development team, communication is achieved using UL. That is the common base where both the analysis model and code model pick their terms and concepts. The UL is constantly evolving fed by both business teams and development teams. The same term or concept should not be mentioned within the analysis model and in the code model under different names. Concepts and terms that are discovered at a coding level are replicated to the UL**,** if the business reveals hidden concepts at the analysis model level; this insight is fed back into the UL too, therefore into the code model. This is the key for domain experts and development teams to evolve the model in collaboration**.** (Millet & Tune, 2015)

At this stage we understand that DDD and Model-driven design are complementary, DDD cannot be achieved without constructing models with business teams and developers' teams. Since they allow the deep understanding of the domain to reach all the way from the minds of domain experts and programmers, through their ubiquitous language, through the software itself, even to the end user — and all the way back again (Evans, Domain-Driven / Model-Driven, 2004).

As a result of applying strategic patterns we end up with core parts of the system identified and isolated so that they can be invested in and evolve independently, big problem domain is split up into subdomains for easier understanding, a model corresponding a specific bounded context is created for each subdomain, each context of the entire system communicate through well-defined interfaces, models are isolated from ambiguity and corruption, when a modification within a specific model does happen, it does not have an impact on other models, code models, often referred as domain models, are isolated from infrastructure code: where the complexity related to technical issues and external libraries are placed, in order to avoid merging technical issues with business concerns.

[6] Source : https://martinfowler.com/bliki/BoundedContext.html

Tactical Design:

This is the implementation part where the bounded contexts are implemented following specific software architecture. Fortunately, DDD relies on a set of patterns that serve the purpose. They help manage the complexity in the solution by shaping the most appropriate architecture for the application. In a nutshell, they help implementing what has been done during the strategic design stage.

After having defined the bounded contexts, each corresponding to an analysis model that will be translated to a code domain model, they are usually shipped into a micro-service like Application Architecture. The fact is that DDD is a 15 Years old approach, but it is expanding only now because of micro services (Boissinot, 2019). That's why people use to say "The DDD is the keystone of Micro-services". A micro-service is a context related to a business use case, which will be materialized by a domain model, which in turn represents a domain boundary (as achieved during the strategic design stage). Domain models are then related to their specific data management system (Database, flux …).

The resulting DDD micro-service application will be then a set of independent micro-services. Micro services advantages are flexibility, maintainability, scalability and the capacity to be Polyglot.

The key takeaway is the fact that we must have one business bounded context corresponding to one micro-service, to one domain model and to only one data source as illustrated in Figure 11.

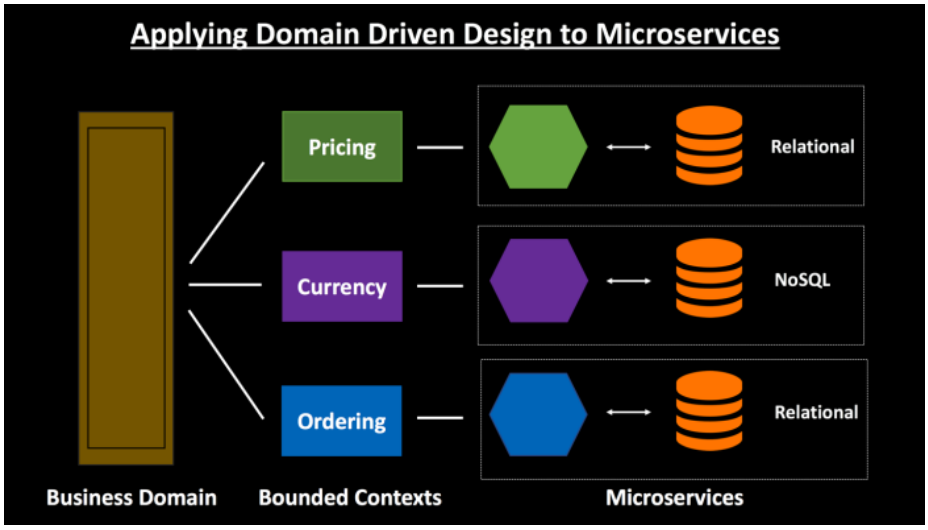


Figure 11 : One business bounded context, one micro-service, one domain model, one database [7].

[7] Source : https://dzone.com/articles/breaking-the-monolithic-database-in-your-microserv

The DDD relies also on what are depicted as clean coding patterns, namely, high cohesion - low coupling, single responsibility principle (SRP), hexagonal architecture, inversion of control and dependency injection principle. This paper will focus on an implementation with a hexagonal architecture.

A hexagonal Architecture, as illustrated in Figure 12 is defined at its center by a hexagon, the biggest part, the most important one, representing the application core; agnostic to any dependency, everything that it needs is defined within this same layer where we put the best developers: This is the DOMAIN layer. The hexagon is then protected by a layer named after APPLICATION layer forbidding the domain layer to be polluted. On this protection, will be plugged subsequent areas such as the INFRASTRUCTURE LAYER, linking the app to databases or any data management systems (flux, RDMS, NoSQL, DBMS …)

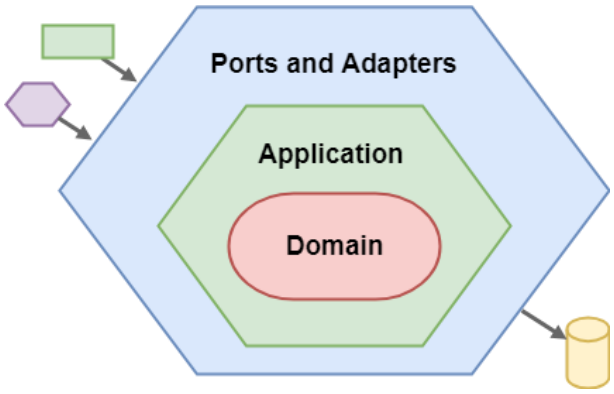


Figure 12: Overview of Hexagonal architecture[8].

Jeffrey Palermo introduced an extension of the hexagonal architecture: The Onion Architecture.

At the center of the onion, resides the domain layer where have been defined Domain services, Domain models and Repositories. Domain Services contain the core functions of the system that delivers value to the business. Domain models are just code models corresponding to the analysis models defined during the strategic stage. Domain models do not reflect how the data are managed by the Data management system, the Data model, included in the infrastructure layer is. Repositories are interfaces describing how data are retrieved from the data management system which can be a database, a JSON file or any other data source.

Around the domain layer, resides the application layer, within which domain services are injected via the dependency inversion principle (DIP) in order to be orchestrated and served to upper layers.

[8] Source: https://github.com/cwoodruff/ChinookASPNETCoreAPIHexArch

Others layers are above the application layer, they are DIP injected with items from both domain layer and application layer. They are the Infrastructure layer and the exposition layer.

The Infrastructure one defines the implementations of the repositories/interfaces described in the domain layer. It also defines all services interacting with items beyond the application scope (DB, rest request, external API calls …) and data models corresponding reflecting how data are saved in the Data source. Every technical related logic resides here, away from the domain logic of the application. This is how the corruption of the business logic with technical concerns is avoided.

The exposition layer is exposing APIs to the outside of the system, to clients such as web applications, mobile applications and third-party applications. The exposition model reflects how data are exposed to API clients, i.e. how clients must consider handling data returned by the API calls.

Even though the models defined at each layer are not intended to reflect each other, they may need to be mapped to each other; Therefore, Adapters need to be defined to do the mapping. Examples of mappings can be the mapping between Domain model and Exposition Model or the mapping between Data model and Domain model.

The whole architecture is summarized in Figure 13.

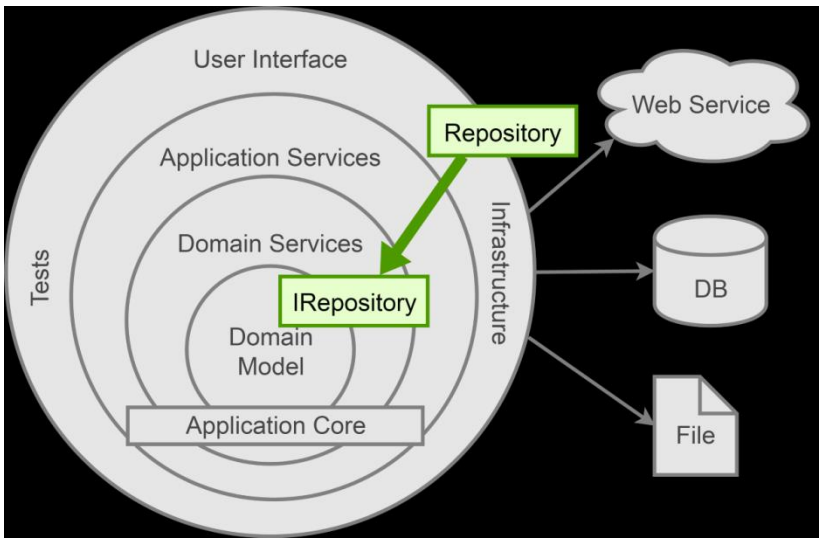


Figure 13: Extended Hexagonal Architecture: The Onion Architecture[9]

How does the DDD bring answers to fined grained questions?

Strategic patterns shape the solution and tactical patterns implement a rich domain model that has been defined. The Domain-driven design does not only focus on the knowledge of the subject (Collaborative Design), but give us useful ways (Strategic Design) of understanding

[9] Source : https://dzone.com/articles/stem-in-onion-architecture-or-fallacy-of-datanbspl

the subject matter, as well as implementing maintainable and scalable software (tactical designs).

2.3.3) Drawbacks of DDD

DDD has been first evoked by Eric Evans in 2004 but it is just on 2013 that we have started noticing activities around that term, with the venue of micro-services.

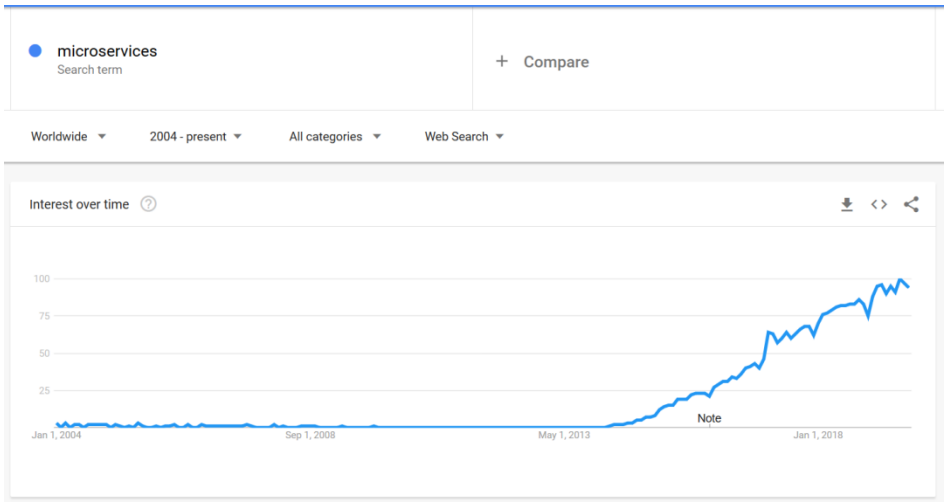


Figure 14: Micro-services searches trend since 2004[10]

This is also because it comes with some drawbacks. DDD Take a lot of effort to implement, moreover, domain expert are expensive to hire, it is very easy to do DDD wrong: Unfortunately, many teams try to implement the DDD in the wrong way by only focusing on tactical patterns; which leads to skipping the DDD step of "distilling the domain" (part of the strategic pattern). Therefore, Sub-domains are not well defined or are not defined at all; domain boundaries are then ignored, the core domain on which the best developers and resources should be involved is unknown, the ubiquitous language that is mandatory for the correlation between analysis model and code model is not defined; because domain boundaries are not clearly defined, software code falls back in what DDD is trying to fight against: the Big Ball of Mud (BBOM) – an anti-pattern where the code parts are mixed and extremely hard or almost impossible to maintain over iterations. Because the UL is not defined, code model evolves during iteration without a feedback loop to the analysis model, and developers start diverging from what have been defined by the domain experts as needs at the beginning. With domain divergence, Software works but not for what it is intend to, leading to huge amount of money lost.

[10] Source : https://trends.google.com/trends/explore?date=all&q=microservices

2.4) Conclusion

On our way to find means to tackle the problem of improving the designing large-scale software, we have been studying three existing approaches in order to get some clues out of them and rely on it to suggest enhanced approaches. Each of the approaches addresses some specific subjects matter more than the others. However, the main outcomes, with respect to the coarse and fined grained questions are highlighted in the following Table 2.

Table 2: Existing approaches with respect to the research problem

Research problem	Coarse Grained Issues	Fine Grained Issues	Existing solutions		
			MDA	BDD	DDD
How can companies improve their way of designing and Developing complex Software?	**How to decide what to build and how to build to provide critical services to thousands of users?** **Designing issue**	**How to educate the teams and let them notice the importance and the priority of getting aligned with the business?**			
		How to speak to the business people and to captivate their interest?			**Collaborative Design:** **Event Storming**
		How to get the most correct domain knowledge as possible?	**Platform Independent Models**	**Ubiquitous Language**	
	How to build and correct quickly what have been decided? **Implementation and maintenance issue**	**Which development pattern helps the team to focus on domain issues as described by domain experts?**		**Automated Tests**	**Strategic Design:** **Bounded contexts**
		How to separate technical implementation concerns from domain logic issues?	**PIM to PSM**		**Tactical Design:** **Hexagonal & Onion architecture**
		How to apply an architectural style facilitating scalability and features enhancements?			

The blank holes in the Solution pane of the table means that despite the usefulness of the existing solutions, there are elements of our research problem that still need to be addressed. In order to suggest solutions, we will proceed to a research methodology relying on surveys and feedbacks.

3. Research Methodology: Qualitative analysis

3.1) Methodology

A qualitative research is a kind of research that produces and analyzes descriptive data, such as written or spoken words and observational behavior of people. It focuses on the "why" rather than the "what" of social phenomena and relies on the direct experiences of human beings as meaning-making agents in their everyday lives (University of Utah)**.** The researcher is interested in knowing the factors conditioning a certain fact in a reality. The qualitative research is usually used to detect needs, make choices and decisions, improve an existing system and test scientific hypotheses. The qualitative research is conducted using methods such as storytelling, observations and interviews. Interviews can be opened, structured or semi-structured. In open interviews, the interviewer does not ask questions that could reorient the interview, the interviewed just have to tell its speech, meanwhile in structured one, the interviewer ask closed questions that have been prepared and planned in a specific order; and in semi-structured interviews, the interviewer prepare some questions in order to keep the speech in a specific lane.

Semi-structured interviews have not only been used in order to confirm hypotheses, but to discover new insights that are unknown as well.

3.1.1) The questionnaire and the interview

Semi-structured interviews have been conducted on a set of seven diverse profiles that have been involved for many years in the software development lifecycle. They are four men and three women, product owners, developers, architects, business analysts, tech leaders and product managers. Their answers where not exclusively related to their current working position but from their previous experiences as well.

As the profiles needed to feel free and not influenced, we did individual interviews. The questionnaire was structured into themes and subthemes.

Context: In order the put the interviewed in confidence, we had them talking about what position they are occupying and their corresponding tasks. This part was specifically supposed to bring to light what kind of stakeholder he/she is expected to be compared to the real set of activities that he/she does play in the software development lifecycle on daily basis.

Knowledge about the designing of large-scale software: The aim here was to get an idea of the knowledge they have about common software designing approaches.

Usage of software and designing approaches: Getting to know if they use their knowledge or are implicated in the software development lifecycle with respect to the approaches, were what mattered here.

The themes Approaches satisfaction feedback and Issues and future expectations were aimed to gather the needs of the interviewed.

The full interview guide is attached to the appendices.

3.1.2) Organization of the result grid (themes and sub-themes)

The interviews have been audio recorded in order to be transcripted by writing later. The result grid have been structured, on one hand into theme and subthemes, and per interviewed on the other hand. The occurrence and frequency of answers according themes has been recorded for analysis purposes.

The full result grid is attached to the appendices.

3.2) Analysis

Both horizontal and vertical analysis could be done. The latter is focused on the interviewed rather than a theme, meanwhile the previous is focused on the themes rather than profiles. Therefore, it is naturally that the choice was the horizontal as the aim of his paper is not based on individuals but on generic facts.

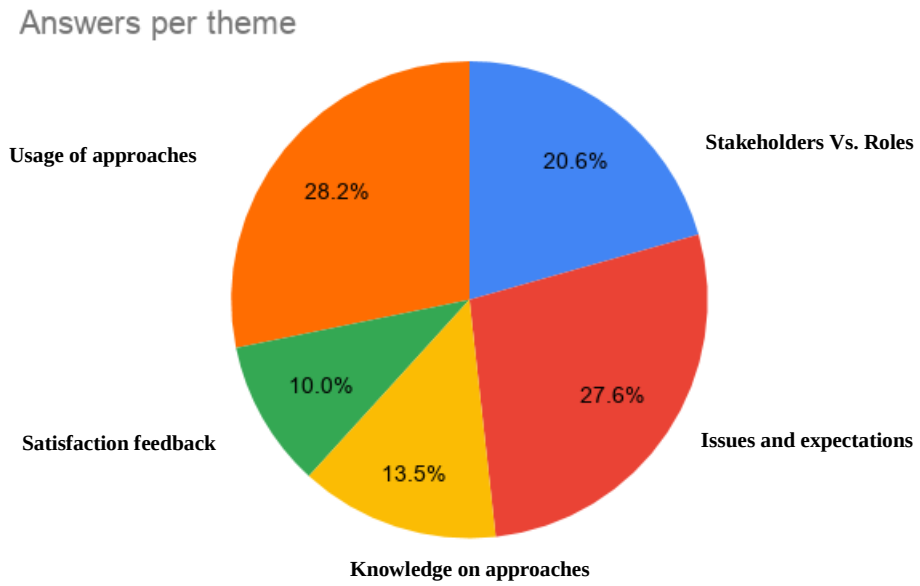


Figure 15: What interviewed where likely to talk about

Let's go further and analyze theme by theme.

3.2.1) Stakeholder's roles vs. activities:

This theme has two subthemes: Expected role and Current activity. Each of the subthemes can have the value Product owner, tech leader, product manager, architect, business Analyst or domain expert.

Here we notice that more than the half of them has or have already had only one expected role but is/was achieving the activities of more than one role such as tech leaders being the architect too and vice versa, developers playing the roles of business analysts and domain expert, product manager being business analyst.

We also notice that there are some profiles who talk much more about what are their roles and their activities namely the business analysts, the tech leaders and the Architects.

3.2.2) Knowledge on approaches

42% of the answers where about Ad hoc knowledge – not specific. The Domain Driven Design methodology pops up with 35% of answers, followed by the Behavior Driven Design with 14% and the Model Driven Approach with 8%.

3.2.3) Usage of software designing approaches

0% has never been implicated or used. 46% acknowledge the necessity of using the Domain Driven Design, 12% for the BDD and 15% for the MDA. Nevertheless only 4% implied a real usage of DDD with 11% mentioning the drawbacks, meanwhile 18% of answers implied real usage on BDD with almost no mentions of drawbacks, finally almost no experience on MDA and 4% of mentions about eventual drawbacks.

3.2.4) Approaches satisfaction feedback:

Very little said here, the respondents were not open to this discussion about this subject Nevertheless, people reported that BDD enhances requirements elicitations and that DDD enhances the value delivered to the client.

3.2.5) Issues and expectations:

This is one of the subjects that has been discussed the most as the profiles were giving their suggestions to the issues they have been going through across their past and present experience.

- About better collaboration between stakeholders: 34% of the answers highlighted an organization issue and where suggesting solutions; meanwhile 22% were complaining about domain knowledge accessibility and another 21% suffered from the gap between the technical language and business one.
- Only 4% of answers mentioned a problem about coding practices
- 19% raised the issue of roles vs. activities.

4. Discussion

4.1) Key Takeaways from the analysis

As Figure 3 shows, and from the theme Stakeholder's roles vs. activities, the profiles where most likely to talk about how they use approaches, how they are involved in the process, their roles, their activities and their expectations rather than the feedbacks on the usage; this leads us to think that the responsibilities are not quite defined, not quite known nor applied. There is a need to break the Silos and switch from project mode to product mode.

From the theme Knowledge on approaches, we understand that the standards approaches are getting applied according to team/company/entity context and that practices from MDA, BDD and DDD can be combined in order to come across the drawbacks of each other.

From "Approaches and satisfaction feedback theme", we noticed that there is a lack of knowledge management about previous experiences with previous approaches. People do not gather enough what went wrong, what to do to avoid, what went right nor how to enhance.

Finally, a better interaction between business team and developer teams and a clear separation of responsibility in the whole chain is necessary.

These new insights, that are complementary to the one we already had from the state of the art, are summarized in Table 3 with respect to our fined grained issues. We will then rely on them in order to provide some guidelines on enhancing the development of large-scale software.

Table 3: Research Summary

Research problem	Coarse Grained Issues	Fine Grained Issues	New insights	Existing solutions		
				MDA	BDD	DDD
How can companies improve their way of designing and Developing complex Software?	**How to decide what to build and how to build to provide critical services to thousands of users?** **Designing issue**	**How to educate the teams and let them notice the importance and the priority of getting aligned with the business?**	Break the silos,			
		How to speak to the business people and to captivate their interest?	From project mode to product mode,			**Collaborative Design:** **Event Storming**
		How to get the most correct domain knowledge as possible?	Manage knowledge and feedbacks	**Platform Independent Models**	**Ubiquitous Language**	
	How to build and correct quickly what have been decided? **Implementation and maintenance issue**	**Which development pattern helps the team to focus on domain issues as described by domain experts?**	**MDA+BDD+DDD**		**Automated Tests**	**Strategic Design:** **Bounded contexts**
		How to separate technical implementation concerns from domain logic issues?	**MDA+BDD+DDD**	**PIM to PSM**		**Tactical Design: Hexagonal**
			MDA+BDD+DDD			

		How to apply an architectural style facilitating scalability and features enhancements?				**& Onion architecture**

4.2) Guideline proposal

Based on Table 3 I present a best guideline proposal for enhancing the designing and development of large-scale software. This guideline is separated in three phases:

- Preparing the organization
- Setting up the knowledge management system
- Designing and developing

The first two steps, while not being explicitly linked to the software industry practices, are still important for avoiding some issues during the third phase that consist of software applying design patterns and principles.

4.2.1) Preparing the organization

Many issues are raised for organizations that are still running on SILO mode, this is an architecture extremely hierarchical creating a black box between the business and the IT. The SILO mode leads to poor collaboration between business and IT as well as role identity and activities issues: such as business people presenting solutions to the IT teams instead of needs.

What is important here is to have within the same micro-team people with clearly delimited roles, that can contain domain experts as well as people from the development team.

In the digitalized era, the common pattern to solve this is to break things up, allow decentralization: going from pyramidal architecture to atomic style architecture with many atomic entities composed of both business teams and IT teams collaborating and working every day as being part of the same team, the same product, the same spirit as illustrated in the new model of IT in Figure 16, and Figure 17 where we have stakeholders in a team at the same hierarchy level.

This is done in as follows:

1) Help everyone understand the common vision and goals:

Collaborators must understand what their roles in the organization objectives are. As well as it is important for individuals to understand how other individuals and teams contribute. This encourages team members to think of departments as links in a chain, instead of as separated silos. A helper practice can be to display the organization goals and metrics, charts, graphs, and other visuals front and center for everyone to see.

2) Create cross-functional teams by encouraging cooperation between departments,

3) Collocate teams physically,

4) Clarify roles.

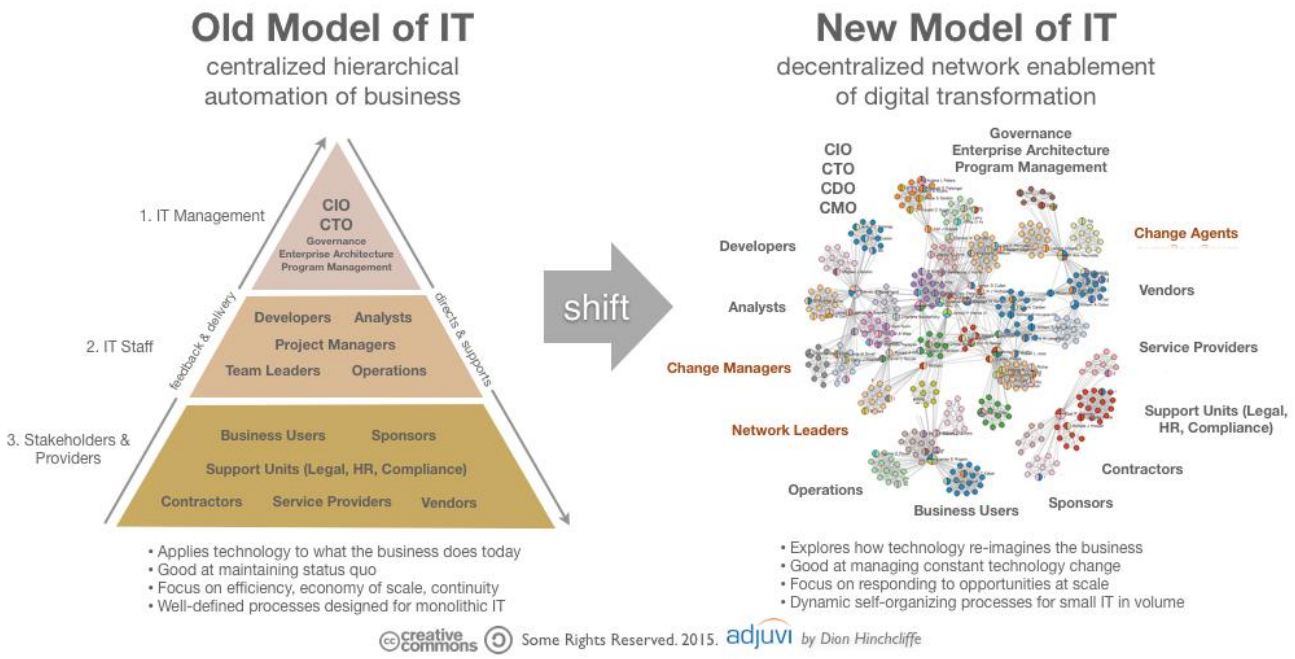


Figure 16: From pyramidal architecture to decentralized architecture[11]

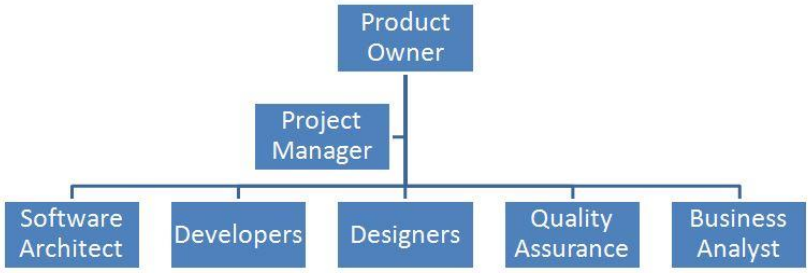


Figure 17 Stakeholders at the same organization level

4.2.2) Setting up the Knowledge Management (KM) System

The analysis results pointed out that there were not enough data on past experiences; nevertheless, they did have some experiences on the evoked design approaches. So, they did not take advantage of their past experiences by creating, updating and controlling documentation about what went wrong? Why? How to correct? And how to avoid? In order to avoid the same errors to keep repeating, there must be a system to facilitate the conservation and above all the sharing of knowledge distributed throughout the company. That is a Knowledge management system.
The concept aims to identify, enhance and disseminate the company's knowledge.

[11] Source : https://dionhinchcliffe.files.wordpress.com/2015/05/old_it_versus_new_it_networks_of_change_agents_enablement.png

This operation is done by:

- Gather the knowledge management infrastructure informations:
  Here, the information about the organization culture (relationships, activities, behaviors at work), the organization hierarchy and structure, the IT Infrastructure (intranet, security, servers) and the physical environment must be gathered,
- Based on the KM infrastructure information, extract the KM activities (Such as learning by observations, face to face meetings, training workshops, editing documentation or wikis …) that are done in order to discover, capture and share knowledge (KM Mechanisms), and the software tools (Such as Confluence, enterprise social networks …) used for the same purpose (KM Technologies),
- Then, a diagnosis must be conducted with respect to the mechanism and the technologies. The KM Mechanism and Technologies must be equally allocated to the discovering, the capture and the sharing of the knowledge. If it is not the case then, technology or mechanism must be recommended in order to fill the gaps.

4.2.3) Designing and developing

With the organization structure encouraging individuals/teams to collaborate closely and a KM system ready to handle the Software knowledge lifecycle, teams can start designing and implementing the software.

The research summary in Table 3 showed us that the MDA, the BDD and the DDD while suggesting best practices, still have drawbacks, illustrated as blank holes in the table. Nevertheless, we noticed the use of common patterns and practices between the three approaches, such as the building of a UL, the creation of models that are agnostic to the technology. Leading me to observe a complementarity across the solutions. Hence, I suggest, in the following, a gathering of practices from the three philosophies. It is separated in three phases:

4.2.3.1) Designing with respect to the requirements

This is where the teams collaborate to capture the requirements of the future system, and above all come out with a set of features ready to be implemented by the developers. For that purpose, the team must:

a) Start by creating the ubiquitous language living documentation:
   If the KM System is well defined, that can be done in a shared well structure wiki, fed by developers and domain experts, every time they will agree on a common language term.

b) Train the team on a sample case about Event Storming workshops: before shipping the teams in workshops for requirement gathering, they must be exercise so that they avoid time waste during the real exercise. Event Storming is a flexible workshop format for collaborative exploration of complex business domains. It is delivering a new type of collaboration beyond silo and specialization boundaries (eventstorming.com)**.** The outcomes are displayed on a board with the form of a

timeline frame including the future actors, commands, and events of the future system as illustrated in Figure 18.

c) Now that the domain experts are closely linked to the development team thanks to the organization structure and the fact that the teams are trained, they can practice event storming workshops with less time compared to the time spent usually.

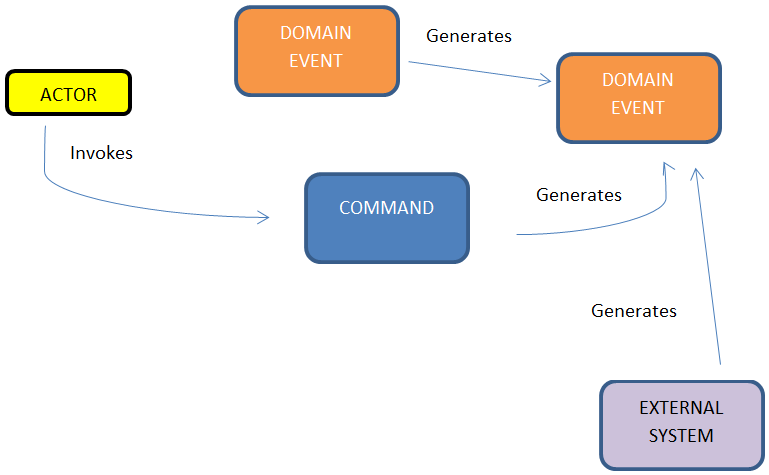


Figure 18: Event storming legend

d) Aggregate the results of the event storming by contexts related to a specific domain. These are the bounded contexts as described by the strategic patterns of DDD (2.3.2))
e) Construct PIM as described by the MDA and document them, then using MDA tools, generate the code models. Leading the team to earn on time, as they are not manually coding the models.

At this stage, we end up with the analysis models (PIM) that are agnostic to the technical platform of implementation, related to specific bounded contexts, the UL documentation is fed from insights from the event storming and the code models corresponding to the bounded contexts. The keynote is the fact that this stage is achieved in less time compared to classical practices thanks to organization structure, trainings and above all the use of MDA tools.

4.2.3.2) Implementing the requirements

This is the where the developers code what have been defined in requirements. This part must be achieved while considering providing value to the business and scalability issues.

a) Design the architecture of the system

Before implementing the business and technical features of the system, I suggest that the developers and architects focus on the technical architecture of the system. As this is going to be the pillars of the maintainability and the scalability of the system. As the micro-service architecture relies on having independent services treating of a specific domain, I hence suggest this architecture and allocate each bounded context to one micro-service. Then within each micro-service, use an onion architecture - as described in the part 2.2.2) (Figure 13) - so that the code implementing the business rules is not polluted by the technical implementation details. E.g.: The code implementing "Submit an order" must be clearly separated from the code responsible of sending requesting the database to register an order – a request being a purely related task. This architecture is summarized in Figure 19.

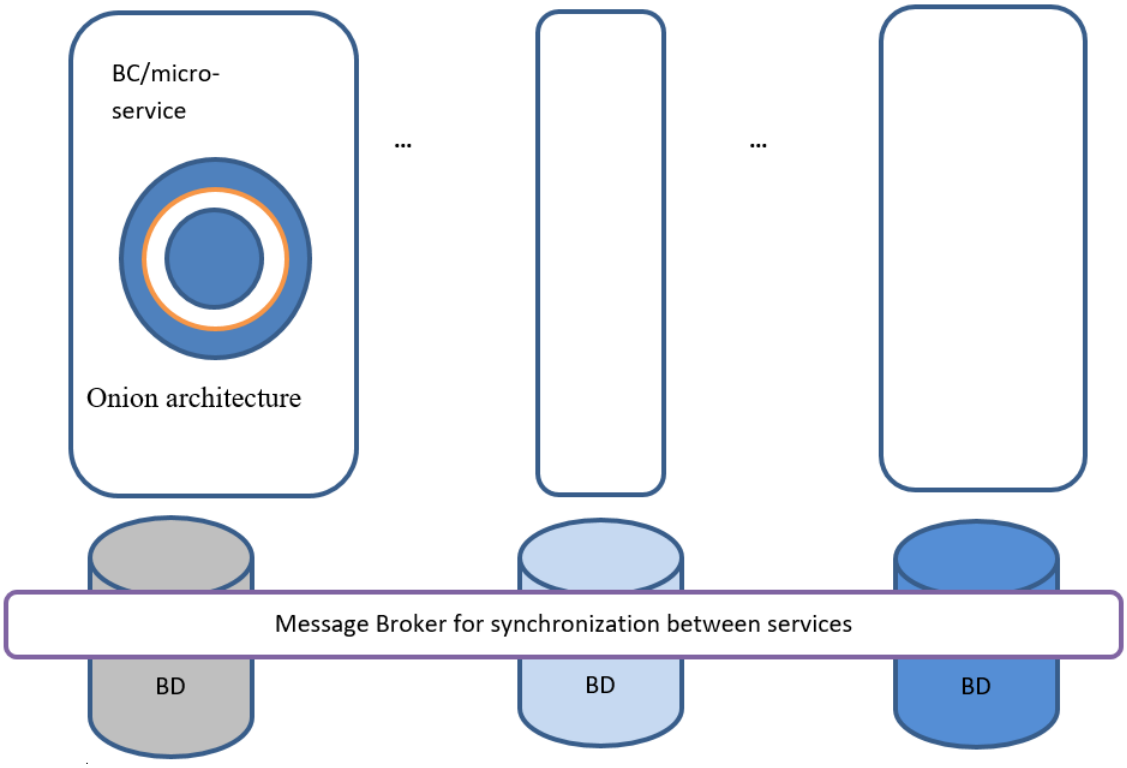


Figure 19: Micro-service coupled to onion architecture

b) Allocating developers roles to specific parts of the architecture:

Considering that we work with a hexagonal architecture (Figure 12), the code is then divided in three general parts, the domain, treating of domain problems, the application orchestrating the business logic and the infrastructure part treating of technical related tasks. I suggest allocating developers as follows: The most complex bounded context problem domains must be treated by the more experienced, talented and passionate developers, while the junior developers with less experience must be allocated to the less complex bounded contexts, application and infrastructure related tasks.

c) Write automated tests to synchronize PIM and code model:

The main issue observed with MDA patterns was the synchronization issue encountered when updating the models. Teams must make sure the implementation code is aligned to the requirements represented by the PIM. With automated tests - as described with BDD (What is BDD?2.2.2) what is BDD?) - every time a modification occurs in the model, the code model is regenerated and validated by running the automated tests, if the tests failed, then the implementation is not anymore aligned to the specifications and must be checked back.

d) Keep the team up to date and aligned with the architecture's principles: The whole architecture of the project can become quite a mess along with the source code increasing. Hence, the developers must keep building up skills to maintain and evolve such architecture. What is suggested here organizing, frequently, a master class for architecture, conducted by an experienced architect, where developers will gather around a table and share their feedbacks about working with the whole architecture of the project, what are the difficulties? Did they come across them and how? Is there any new stuff on the market enhancing working with that architecture? This will lead the developers to remain within a high abstraction mindset and avoid just writing code.

The suggested guidelines for enhancing the making of large-scale software within organizations are summarized in Table 4 for a better visibility.

Table 4: Overview of suggested guidelines

Fine Grained Issues	Guidelines		
	Preparing the organization	Set up KM System (transversal solution)	Designing and Development
How to educate the teams and let them notice the importance and the priority of getting aligned with the business?	Help everyone understand the common vision and goals Clarify roles		
How to speak to the business people and to captivate their interest?	Create cross-functional teams		Trainings on Event Storming
How to get the most correct domain knowledge as possible?	Collocate teams physically		**Ubiquitous Language living documentation** **Event Storming**

Which development pattern helps the team to focus on domain issues as described by domain experts?			**Define Bounded contexts** **Construct PIM then generate code models** **Edit automated behavioral tests**
How to separate technical implementation concerns from domain logic issues?			Design Micro-service over onion architecture
How to apply an architectural style facilitating scalability and features enhancements?			Allocate developers to specific parts of the architecture Keep the team up to date and aligned with the architectures

5. Assessment of guidelines: case of AXA Bank ISD

In order to check if the suggested guidelines can bring valuable answers to the research problem, an assessment is necessary. As an apprentice within AXA Bank ISD, I had the occasion to compare some practices of the guidelines to some of the company strategies in order to observe the results.

5.1) Organization refactoring within AXA Bank ISD

5.1.1) Overview

I had the occasion to assist to the agile refactoring of the department. The aim for the department was to get shipped in a Scaled Agile Framework for Lean Enterprises (SAFe) program.

Being a lean enterprise means delivering value to the end customer with the minimal waste of resources. SAFe is a scalable and configurable framework that helps organizations deliver new products, services, and solutions in the shortest sustainable lead time. It is a system that guides the roles, responsibilities, and activities necessary to achieve a sustained, competitive technological advantage. (SCALED AGILE ,Inc., 2018).

SAFe is the combination of agile with Lean thinking. It introduces the five core competencies of the lean enterprise:

Lean-Agile Leadership, which is about Lean-agile leaders driving and sustaining organizational change and operational excellence by empowering individuals and teams to reach their greatest potential.

Team and technical agility which is about lean-agile principles and practices agile teams need to be high-performing and well-designed technical solutions that support current and future business needs;

DevOps and release on demand: where principles and practices of DevOps are providing the enterprise with the capability to release value in whole or in part, at any time necessary to meet market and customer demand;

Business solutions and Lean systems engineering : set of practices that help building large solutions, involving considerable contributions from external supplier, have significant compliance concerns and demand extensive and broad multidisciplinary skills;

Lean portfolio management describes how an enterprise can implement lean approaches to strategy and investment funding, agile portfolio operation and lean governance for a SAFe portfolio. (SCALED AGILE ,Inc., 2018)

The whole transformation process is beyond the scope of this paper; thus, we will focus on how the Team and Technical Agility competency have been implemented.

The team and technical agility competency is an effective way to assess the first part of the guideline — that is preparing the organization — since it relies, besides others, on the decentralization of decision making and the application of system thinking.

5.1.2) Team and Technical Agility transformation

The implementation of this competency is done around a Program increment (PI) - a time box during which an Agile Release Train (ART) delivers incremental value in the form of working, tested software and systems. An ART is a set of agile teams, which, along with other stakeholders, incrementally develops, delivers one or more solutions supplying value. PIs are typically 8-12 weeks long. The most common pattern for a PI is four development Iterations, followed by one Innovation and Planning (IP) Iteration (SCALED AGILE ,Inc., 2018). Iterations are named after sprints.

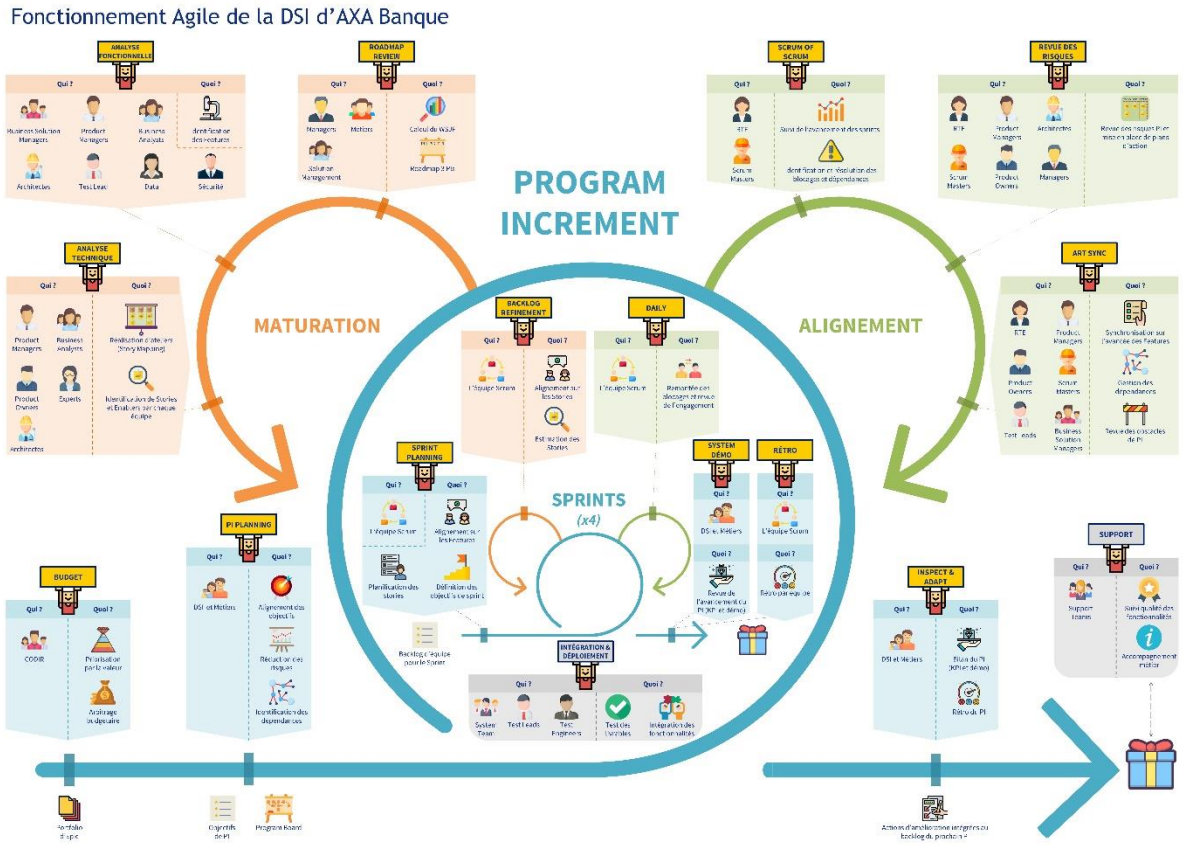


Figure 20: Team and technical Agility Process of AXA Bank ISD

More specifically about AXA bank ISD, the Team and technical Agility transformation is composed of 3 general stages (Figure 20):

- Program Increment
- The Alignment happening during the PI
- The maturation occurring during the PI after the Alignment stage

5.1.2.1) Program Increment

It starts by the definition of its budget by the executive committee, during which, the budget arbitration is done, and the value brought to the whole system is prioritized. Next, happens the PI planning, achieved by the Business People and the ISD teams; here the goals for the whole PI are defined and aligned to the strategy of the bank as well as reducing the risk inherent to those goals. After the PI planning, the execution of the PI can begin with a set of 4 sprints of 2 weeks each.

A sprint starts by a sprint planning stage where the team defines the sprint objectives and backlog - the set of items to be developed by the team. Daily meetings occur each morning in order to give daily updates about what is being developed, achieved or what could be the possible issues. Stuff that are developed during the sprint are tested, integrated, and deployed, this is the continuous integration and delivery of the new features to the previous delivered ones. Before the end of the sprint, teams proceed to a backlog refinement consisting of making sure developments are aligned with what was supposed to be built. At the end of the sprint, occur the system demonstration and sprint retrospective where teams present to the stakeholders what have been developed during the sprint and feedbacks about what went eventually wrong, how to enhance it or avoid.

5.1.2.2) Alignment

As its name highlights, here teams make sure they are still on track with what have been defined during the PI planning, if not, issues are raised and unblocked as fast as possible, so that the ART can continue to go forward. The alignment operation consists of 3 operations, namely SRUM of SCRUM, Risk review and ART synchronization.

The SRUM of SCRUM is done by the Release Train Engineer (RTE) – the one in charge and coaching the ART – with the Scrum Masters, during which, the ongoing diverse springs are monitored, and potential problems are raised and resolved.

The risk review is about the PI risks being analyzed, and actions being set up accordingly. This is done with the collaboration of the RTE, product managers, architects, scrum masters, product Owners and managers.

During the ART synchronization the RTE, the product managers, the product owners, the scrum masters, the test leaders and the business solution managers meet in order to get synchronized on how features are getting developed and treat potential dependency problems between features.

5.1.2.3) Maturation

This stage happens just before the beginning of a new PI. The goals are about: doing a roadmap review of the three previous PIs including the current one; doing a functional analysis of what can be done for the next PI. This is where the feature discovery happens with the help of business solution managers, product managers, business analysts, architects, test leads, data engineers and the security guys; doing a technical analysis which, consists of workshops for identifying new stories and enablers for every team.

At the end of each PI, the ISD teams and domain experts meet in order to show what have been done during the PI and feedbacks on what went eventually wrong, how to enhance it or avoid: this is the retrospective at the PI scale. Finally, when the product is delivered after a PI to the end customer, the support operation begins thanks to the support teams.

5.2) Knowledge Management (KM)

AXA decided to launch the setup of a knowledge management operation in order to take advantage of their past experiences, acquiring new knowledge and avoid the same errors to keep repeating. The operation has started by a diagnosis consisting of an inventory of how we discover, capture, and share knowledge; then actions have been taken to enhance the problem discovered.

Table 5: Knowledge management within the ISD

	DISCOVERY	CAPTURE	SHARING
Technology	Confluence	Confluence	Confluence
Mechanism		Physical training programs/classes, User manuals updates, Creating POCs & White papers	

The Table 5 shows that knowledge Discovery was weakly supported as well as Knowledge sharing. As a correction they have introduced a new program called “Lunch & Learn”: a subject is chosen by a leader for specific Lunch time, and discussions happen on the subject area, people share their knowledge, ideas, thoughts & experiences. The key here is that the workshop is more interactive than the training classes that seem to be college courses with master and students.

5.3) Trainings around event storming and applying the hexagonal architecture

5.3.1) Training Workshops

The training workshops consisted about teams being educated on how to apply the strategic patterns of DDD with real life situations. One of the most practiced activities was the event storming. Figure 21 illustrates the result of one of the training sessions done within the ISD locals.

Figure 21: The result of an event storming

5.3.2) Keep the teams up to date with Coding Sessions

The coding sessions were about developers gathering twice a month for 1 hour to learn around the hexagonal architecture. They have even designed a software base built on top of the Spring-boot Framework for JAVA Enterprise application, with a hexagonal architecture named after SAMA: “Socle applicatif des micro-services chez Axa banque” that is used as a dependency for every project of the ISD. It forces, seamlessly, the developer to code according to the good patterns.

5.4) Key Performance Indicators (KPI) and appraisal

In order to assess the application of the suggested guidelines on the different processes of the ISD, we must first highlight the KPI, namely the number of released features, the commitment achievement rate, the number of released stories – these are subsets of features – the number of released story points – categorizing the complexity level of a story development– the number of features ready for the next PI, the Incident reported rate – that is the ratio between the delivered features and the incident related features – and the number reported bugs in production.

We took a program Increment as the time unit of assessment. The first PI occurred in January, since, three PIs have been achieved over 10 projects that are either done or in progress. The Table 6 shows clearly the delivery enhancements over the PIs.

Table 6: Appraisal of some guidelines within AXA Bank ISD

	AVERAGE BEFORE REFACTORING ON 8 WEEKS	**PI1**	**PI2**	**PI3**
RELEASED FEATURES	50	64	120/238	195/302
RESPECT FOR THE COMMITMENT	NA	NA	57%	65%
STORIES DELIVERED	700	964	1160	1254
RELEASED STORY POINTS	1999	2550	2898	2954
FEATURES READY FOR THE NEXT PI	NA	40	86	90
INCIDENT REPORTED RATE	25%	13%	18%	12%
NUMBER OF BUGS REPORTED	15	13	12	10

6. Conclusion and further work

This paper was about studying how companies can improve how they develop large-scale software with respect to software development methodologies and design patterns; as nowadays more than ever, they are used on a daily basis. For this purpose, it was essential to make sure the research problem was quite understood by, on one hand, distilling the problem into smaller problems and on the other hand, categorizing the problems by design related issues and technical implementation related issues. The state of the art has then been reviewed. This way we discovered that there are already some methodologies, namely MDA, BDD, and DDD that suggest design and implementations practices to solve the issues. Thanks to a qualitative research through semi-structured interviews with stakeholders from diverse backgrounds and companies, feedbacks have been gathered about the existing solutions and complementary information have been discovered about how we can extend the current solution space and provide useful guidelines.

Even though this paper focuses on software development methodologies, the two first parts of the suggested guidelines instigates organization refactoring, and Knowledge management system set up prior to software development practices described in the last part. The latter cannot be applied in a SILO like enterprise architecture; there must be a shift to a decentralized architecture, where teams are not thinking project, but product. Such decentralized architecture promotes teams constituted of diverse stakeholders of the products, from business guys to developers interacting and working together.
The second part of the guidelines combined the practices from MDA, BDD and DDD in order to come across the shortcomings of each other. Especially using MDA principles to generate bounded contexts models and using automated tests suites as described in BDD to keep models aligned with specification, leading to team drastically earning on time.

Finally, in order to be assessed, the guidelines have been assimilated to the recent set of transformations and events that has been happening in AXA Bank where I've been doing my apprenticeship within the ISD as a Java developer. The KPI were delivery oriented – as they focused primarily on the product delivered more than the final value to the business– and shown that the application of guidelines was, at least decreasing the amount of software defects and incidents that the bank was facing.

However, it was not possible to fully assess the suggested guidelines as some of them are just living their early stages within the DSI. Such as the application of the hexagonal architectures, that has started being applied just since less than one year. It is hence complicated to get some useful feedbacks about the brought business value. In addition, assessing the suggested guidelines in only one company is not enough; it must be compared to other practices in other companies.

List of acronyms

ART: Agile Release Train

BBOM: Big Ball Of Mood

BC: Bounded Context

BDD: Behavior Driven Design

CIM: Computational Independent Model

DAO: Data Access Object

DDD: Domain Driven Design

ISD: Information System Department

MDA: Model Driven Architecture

OMG: Object Management Group

PIM: Platform Independent Model

PSM: Platform Specific Model

SAFe: Scaled Agile Framework

SRP: Single Responsibility Principle

TDD: Test Driven Design

UL: Ubiquitous Language

UML: Unified Modeling Language

XMI: XML metadata interchange

Bibliography

Ambler, S. W. (n.d.). *Are You Ready For the MDA?* Retrieved September 17, 2019, from agilemodeling.com: http://www.agilemodeling.com/essays/readyForMDA.htm

Atem de Carvalho, R., Luiz de Carvalho e Silva, F., & Manhaes, R. S. (2010, June). *Mapping Business Process Modeling constructs to Behavior Driven Development Ubiquitous Language.* Retrieved April 29, 2019, from researchgate.net: https://www.researchgate.net/publication/45925747_Mapping_Business_Process_Modeling_constructs_to_Behavior_Driven_Development_Ubiquitous_Language

Bhatti, N. S., & Malik, A. M. (2015, April 10). *Model driven architecture.* Retrieved April 29, 2019, from researchgate.net: https://www.researchgate.net/publication/274916541_Model_Driven_Architecture

Boissinot, G. (2019, 02). What is Domain Driven Design ? (P. SIMO, Interviewer)

Evans, E. (2004, February 04). *Domain-Driven / Model-Driven.* Retrieved from DDD Community: https://dddcommunity.org/uncategorized/evans_2004/

Evans, E. (2004). *Domain-Driven Design: Tackling Complexity in the heart of software),*pp 7-20, 2004.

eventstorming.com. (n.d.). *eventstorming.com*. Retrieved 09 27, 2019, from eventstorming.com: https://www.eventstorming.com/

France, R., & Rumpe, B. (2007, June). *Model-driven Development of Complex Software.* Retrieved April 29, 2019, from researchgate.net: https://www.researchgate.net/publication/4250888_Model-driven_Development_of_Complex_Software_A_Research_Roadmap

Millet, S., & Tune, N. (2015). *Patterns, Principles, and Practices of Domain-Driven Design,* Indianpolis: Wrox, pp 4-70, 2015.

North, D. (2006, March). *Introducing BDD.* Retrieved April 29, 2019, from dannorth.net: http://dannorth.net/introducing-bdd

Object Management Group. (2001). *MDA Specifications.* Retrieved September 25, 2019, from OMG: https://www.omg.org/mda/specs.htm

SCALED AGILE ,Inc. (2018). *SAFe® 4.6 Introduction.*

Solis, C., & Wang, X. (2011, October). *A Study of the characteristics of Behaviour Driven Development.* Retrieved April 29, 2019, from researchgate.net: https://www.researchgate.net/publication/224265781_A_Study_of_the_Characteristics_of_Behaviour_Driven_Development

Thomas, D. (2003, January). *UML - Unified or Universal Modeling Language?* Retrieved September 16, 2019, from www.jot.fm: http://www.jot.fm/issues/issue_2003_01/column1/

Truyen, F. (2006, January). *The Fast Guide to Model Driven Architecture - The Basics of Model Driven Architecture.* Retrieved from www.omg.org: https://www.omg.org/mda/mda_files/Cephas_MDA_Fast_Guide.pdf

University of Utah. (n.d.). *What is Qualitative Research.* Retrieved from Health University of Utah: https://nursing.utah.edu/research/qualitative-research/what-is-qualitative-research.php

Wanivenhaus, J. (2017, October 05). *Behavior-Driven Development: The Origins.* Retrieved September 17, 2019, from DZone: https://dzone.com/articles/behavor-driven-development-the-origins

Appendices

Semi-structured Interview Guide

Guide d'entretien:

Thème: Concevoir et développer des logiciels modernes avec efficacité et efficience.

Consigne:
Etudiant en dernière année de Master Management de Système d'Informations et de connaissances,

Je fais une étude sur comment améliorer, la conception et le développement des logiciels modernes (à plusieurs milliers d'utilisateurs, et à domaine complexe tels banque, assurance, commerce digital ...).

Je mène des interviews avec des parties prenantes de la fabrication de logiciels (Product Owner, Business Analysts, Tech Leader, développeurs et sachants métier) dans le but de recueillir leurs impressions quant aux pratiques actuelles, les problèmes rencontrés et leur besoins. Afin de pouvoir mener une étude croisée et en ressortir des potentielles solutions.

Sous thème 1: Contexte

1. Parlez-moi de votre métier?

2. Intervenez-vous/avez-vous déjà intervenu d'une quelconque façon, durant le cycle de vie (la conception ou la fabrication ou l'utilisation) d'un outil logiciel censé aider à atteindre les objectifs de l'entreprise ?

3. Quel rôle y jouez-vous ?

Sous thème 2 : Connaissance des approches de conception logicielle

4. Quelles pratiques/méthodes/approches de conception/développement logiciel connaissez-vous? particulières durant le cycle de vie ?
   S'ils ne citent pas l'un de ces éléments: MDA, BDD, DDD passer au 5, sinon passer au 6

5. Avez-vous entendu parler des approches de conception logicielles suivantes:
   MDA
   BDD
   DDD ?

6. En quoi consistent généralement les méthodes MDA, BDD, DDD ?

Sous thème 3 : Utilisation des approches de conception logicielle

7. Avez-vous déjà appliqué (été impliqué dans) une de ces approches lors de vos précédentes interventions ?

8. Qu'en est-il de votre intervention actuelle ? êtes-vous impliquez? L'appliquez-vous ? *Si non poser la 9*

9. Pourquoi ne l'appliquez-vous plus ?

Sous thème 4: Retour d'expérience d'utilisation des approches de conception logicielle

10. Mettre les pratiques/méthodes/approches actuelles en exécution améliore-t-il la qualité du logiciel produit?
    - PO et BA:
    Améliore-t-il la collaboration avec les sachant métier ?
    Améliore-t-il la découverte, la description , l'analyse et la validation des fonctionnalités ?
    - Tech leader et développeur**s** :
    Améliore-t-il la collaboration avec les PO et BA?
    Améliore-t-il la qualité, lisibilité, du code, réduisent-ils la difficulté de développement et de maintenance, les bugs en production ?
    - Sachant métier: Retrouvez-vous vos spécifications dans le produit livré ? La qualité du service fournit par le logiciel est-elle meilleure?

11. Selon vous, les pratiques actuelles sont-elles évidentes, pas assez évidentes ou difficiles à mettre en œuvre ? / les façons de procéder actuellement vous conviennent elles ?

Sous thème 5: Attente concernant les approches de conception logicielle

12. Pouvez-vous me décrire ce que vous attendez des telles pratiques qui vous rendrait plus productif? Qui vous aiderait davantage dans votre rôle de partie prenante ? Qui produirait de meilleurs logiciels?

Qualitative analysis summary

qualitative_analysis

Analysis by frequence of occuring

THEME	SUBTHEME	SUB-SUBTHEME	INTERVIEWED 1	INTERVIEWED 1	INTERVIEWED 2	INTERVIEWED 2	INTERVIEWED 3	INTERVIEWED 3	INTERVIEWED 4
"Stakeholder role & activities"	Expected Role	Product Owner							
		Tech Leader	"Lead technique"	1	"Je suis Tech leader"	1			"Business Analyst","je
		Product Manager							
		Architect							
		Business Analyst							
		Developer					"concepteur développ	1	
		Domain expert							
	Current Activity	Product Owner							
		Tech Leader	"conception technique","	2	"Garantir la qualité du	1			
		Product Manager							
		Architect	"Valider l'architecture de	1					
		Business Analyst					"il y a des choix qui so	2	"J'interviens sur la ma
		Developer					"concepteur développ	1	
		Domain expert					"on était obligé de déf	1	
Percentage				18.18%		6.45%		16.13%	
Knowledge about desiging complex softwares	ne stuff about software dev lifecycle(AD HOC)		"Maturation en analyse f	4	"Tavailler sur la définit	3	"Chacun se construit s	1	
	Desinging approaches and patterns	MDA							
		BDD	"Projet DSP2 où BDD y	2	BDD by Design	1	"Moi j'ai fait du TDD"	1	
		DDD	"Nous connaissons tous	1			"là j'ai découvert le DD	1	
Percentage				31.82%		12.90%		9.68%	
Usage of software desiging approaches	Never used/implciated								
	MDA	Acknowledge necessity			"On va passer par cet	2			"oui forcément je vois
		Has experience							
		Drawbacks							
	BDD	Acknowledge necessity	"Validation des besoins	1	"inclure réellement les	4			
		Has experience	"Projet DSP2 où BDD y	2			"on fait du Bdd"	1	
		Drawbacks							
	DDD	Acknowledge necessity	Y a pas mal d'échanges	1	"Ce n'est pas encore t	5	"J'ai fini par l'adopter"	4	"ça englobe tout le mo
		Has experience		1	"C'est encore trop peu	1			
		Drawbacks					"c'était contraignant p	3	
Percentage				22.73%		38.71%		25.81%	
Approaches Satisfaction feedback	MDA	Feedback not relevant enough							
		Enhanced software overall quality							
		Enhanced collaboration between dev teams and domain experts							
		Enhanced requirements elicitation							
		Enhanced code quality and maintainability							
		Enhanced value delivered to the business							
		Straightforward							
		Abstract							
	BDD	Feedback not relevant enough							
		Enhanced software overall quality							
		Enhanced collaboration between dev teams and domain experts							
		Enhanced requirements elicitation							
		Enhanced code quality and maintainability							
		Enhanced value delivered to the business							
		Straightforward							
		Abstract							
	DDD	Feedback not relevant enough			"On n'a pas encore d	1			
		Enhanced software overall quality					"ça permet de stand	4	
		Enhanced collaboration between	Y a pas mal d'échange	1			"tu commences par	1	
		Enhanced requirements elicitation					"Tu sais tu sais te pr	2	
		Enhanced code quality and maintainability					"t'es pas obligé de c	1	
		Enhanced value delivered to the business			"les développeurs es	2			
		Straightforward							
		Abstract							
Percentage				4.55%		9.68%		25.81%	
Future expectations/issues	Better collaboration with stakeholders	Organisation hierarchy decentra	"Pas de visibilité sur l'éc	2	"C'est encore très hi	4			
		Domain Knowledge accessibilty	"Comprendre les besoins	1	"on n'a pas forcémer	2	"le métier ne sait pas	3	
		Shared language	"Barrière de langue entre	2	"À un même terme o	4			
	Coding practices			0					
	Roles vs activities			0			"les métiers doivent	4	"c'est que tout le mo
Percentage				22.73%		32.26%		22.58%	
Total				22		31		31	

qualitative_analysis

INTERVIEWED 4	INTERVIEWED 5	INTERVIEWED 5	INTERVIEWED 6	INTERVIEWED 6	INTERVIEWED 7	INTERVIEWED 7	Cumulated Frequencies	
							Absolute	Relative
		1			"JE suis product Own	2	3	2%
2							4	2%
			"je suis Coté métier"	1			1	1%
	"mettre en place des	1					1	1%
							0	0%
							1	1%
							0	0%
					"Prioriser les sujets et	1	1	1%
	"je mets en place des	2					5	3%
			"tout le volet métier p	3			3	2%
	"j'ai une casquette arc	4					5	3%
3			"je dois faire des expr	2	"Je suis tout le temps	2	9	5%
							1	1%
							1	1%
31.25%		23.53%		30.00%		31.25%	35	21%
			"plusieurs ateliers de	1	"Une fois qu'on a iden	1	10	6%
	"mais parce que juste	1					1	1%
							4	2%
	"il voir le DDD comme	5	"Entendu parler d'une	1			8	5%
0.00%		17.65%		10.00%		6.25%	23	14%
							0	0%
4							6	4%
							0	0%
	"c'est moins adapté m	2					2	1%
							5	3%
	"j'étais sans arrêt en a	2			"Tu mets en place un	3	8	5%
							0	0%
3	"Un des intérêts du D	4	"j'aimerais bien savoir	3			20	12%
							2	1%
			"Je ne pense pas que	2			5	3%
43.75%		23.53%		25.00%		18.75%	48	28%
							0	0%
							0	0%
							0	0%
							0	0%
							0	0%
							0	0%
							0	0%
							0	0%
							0	0%
							0	0%
							0	0%
					"Oui clairement oui	**1**	1	1%
							0	0%
							0	0%
							0	0%
							0	0%
							1	1%
							4	2%
	"d'interagir avec le r	**2**					4	2%
							2	1%
							1	1%
	""en adéquation par	**2**					4	2%
							0	0%
							0	0%
0.00%		**11.76%**		**0.00%**		**6.25%**	17	10%
0	**"Je pense que on a**	**5**	**"d'identifier des gen**	**1**	**"Le nombre de réuni**	**3**	15	9%
	"sensation de toujou	**1**	**"pas assez de perso**	**1**	**"De la documentatio**	**2**	10	6%
			"le métier va s'expri	**5**			11	6%
	"équipes de dévelop	**1**			**"On ne valorise pas**	**1**	2	1%
4	**"beaucoup de chos**	**1**					9	5%
25.00%		**23.53%**		**35.00%**		**37.50%**	47	28%
16		34		20		16	170	100.00%