]Designing complex softwares.

PLAN

[Table of illustrations 2](#_Toc20385593)

[Table of tables 2](#_Toc20385594)

[Abstract: 3](#_Toc20385595)

[Introduction: 3](#_Toc20385596)

[I. Research problem 3](#_Toc20385597)

[A) Problem description 3](#_Toc20385598)

[B) What is Design, develop, complex software? 4](#_Toc20385599)

[C) Fined grained view and sub categories of the problem 4](#_Toc20385600)

[II. State of the art 5](#_Toc20385601)

[A) Model Driven Architecture (MDA) 5](#_Toc20385602)

[i) History 5](#_Toc20385603)

[ii) What is MDA? 6](#_Toc20385604)

[iii) Drawbacks of MDA 8](#_Toc20385605)

[B) Behavior Driven Design (BDD) 8](#_Toc20385606)

[i) History 8](#_Toc20385607)

[ii) What is BDD? 9](#_Toc20385608)

[iii) Drawbacks of BDD 13](#_Toc20385609)

[C) Domain Driven Design (DDD) 13](#_Toc20385610)

[i) History 13](#_Toc20385611)

[ii) What is DDD? 14](#_Toc20385612)

[iii) Drawbacks of DDD 20](#_Toc20385613)

[D) Conclusion 21](#_Toc20385614)

[III. Research Methodology: Qualitative analysis 23](#_Toc20385615)

[A) Methodology 23](#_Toc20385616)

[i) The questionnaire and the interview 23](#_Toc20385617)

[ii) Organization of the result grid (Subjects and sub-subjects) 24](#_Toc20385618)

[B) Analysis 24](#_Toc20385619)

[i) Theme repartition 24](#_Toc20385620)

[C) Results and comments 26](#_Toc20385621)

[i) The Organization refactoring 27](#_Toc20385622)

[ii) The knowledge Management 27](#_Toc20385623)

[iii) Demystifying the Domain Driven Design 28](#_Toc20385624)

[IV. Assessment and validation of results: case of AXA Bank ISD 31](#_Toc20385625)

[A) Organization refactoring within AXA Bank ISD 31](#_Toc20385626)

[i) Overview 31](#_Toc20385627)

[ii) Team and Technical Agility transformation 32](#_Toc20385628)

[iii) Key Performance Indicators (KPI) and appraisal 34](#_Toc20385629)

[B) Demystifying the Domain Driven Design 34](#_Toc20385630)

[V. Conclusion and further work 35](#_Toc20385631)

[Table of acronyms 35](#_Toc20385632)

[Bibliography 35](#_Toc20385633)

[Appendices 36](#_Toc20385634)

[A) Semi-structured Interview Guide 36](#_Toc20385635)

[B) Questionnaire for the quantitative analysis 38](#_Toc20385636)

# Table of illustrations

[Figure 1 MDA Overview 7](#_Toc20325697)

[Figure 2 MDA Methodology 8](#_Toc20325698)

[Figure 3: A feature “Is it Friday yet” with steps defined in Gherkins 10](#_Toc20325699)

[Figure 4: How Cucumber Works 10](#_Toc20325700)

[Figure 5: “Is it Friday yet” corresponding steps definition in a java test file 11](#_Toc20325701)

[Figure 6: Behavior Driven Development methodology 12](#_Toc20325702)

[Figure 7: Two bounded Contexts independent from each other 15](#_Toc20325703)

[Figure 8 : One business bounded context, one micro-service, one domain model, one database 17](#_Toc20325704)

[Figure 9: Overview of Hexagonal architecture 18](#_Toc20325705)

[Figure 10: Extended Hexagonal Architecture: The Onion Architecture 19](#_Toc20325706)

[Figure 11: Micro-services searches trend since 2004 20](#_Toc20325707)

[Figure 12: What interviewed where likely to talk about 24](#_Toc20325708)

[Figure 13: From pyramidal architecture to decentralized architecture 27](#_Toc20325709)

[Figure 14: Team and technical Agility Process of AXA Bank ISD 32](#_Toc20325710)

# Table of tables

[Table 1 The research problem distilled 5](#_Toc20385530)

[Table 2 Existing approaches with respect to the research problem 21](#_Toc20385531)

[Table 3 Suggested solutions 29](#_Toc20385532)

# Abstract:

# Introduction:

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

We use complex software platforms on daily basis (Airbnb, Leboncoin, platforms and mobile applications for banking services) and we cannot ignore how vital they are to us. Complex 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 kind. However, we still facing some project failures due to requirements not met, products not easily maintainable, misuse of technologies etc.  
This paper did focus on software development methodologies, software designs patterns with the aim to suggest a set of guidelines in order to enhance the process of developing such softwares. 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.

# Research problem

## Problem description

Companies seek building customer’s loyalty, supply ever better products and services on an ongoing basis. Hence, they face the challenge of creating software for critical domains such as health, finance, e-commerce, loans and mobile banking. Yet, they are constantly facing customer complaints about products, frequent incidents, frequent project abandon in favor of new ones.

The same kind of issues were also encountered within the AXA bank Information System Department (ISD), where, I’ve been participating to the designing of Java based applicative systems providing customer bank services. 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 complex Software?

## What is Design, develop, complex software?

Designing is deciding what to build and How to build. Developing is implementing what have been decided. A complex Software is 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.

## Fined grained view and sub categories of the problem

Getting to know a more fined 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 software 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, that 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 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?** |
| **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.

# 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.

## Model Driven Architecture (MDA)

### 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 Designing UML class Diagrams at the beginning of the project, 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)

### What is MDA?

#### Definition

#### Model Driven architecture is an approach to using models in software development to produce applications independent of the infrastructure they use. (Bhatti & Malik, 2015) The aim is to use one set of models – the Platform Independent Models (PIM) that are independent of technical infrastructure to specify the features of an application, 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. This latter supports four core standards/meta-models 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.

#### Methodology

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 figure1.

Once the PIM have been specified, convert the PIM first 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)***.*** This conversion consists of transforming the PIM to a model, that is linked to a specific programming language, operating system or database that computers can run. This is done by using MDA development tools, inferring platform specific rules, available now from many vendors. An illustration can be to transforming UML class diagram to an Oracle relational Model or to a Plain Old Java Object Model. This second step occurs within the second disk from the center of the MDA overview from figure1.

The whole process summarizing the two steps is illustrated in figure 2.

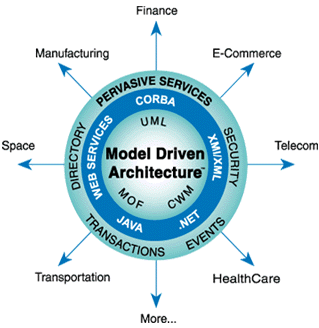


Figure 1 MDA Overview[[1]](#footnote-1)

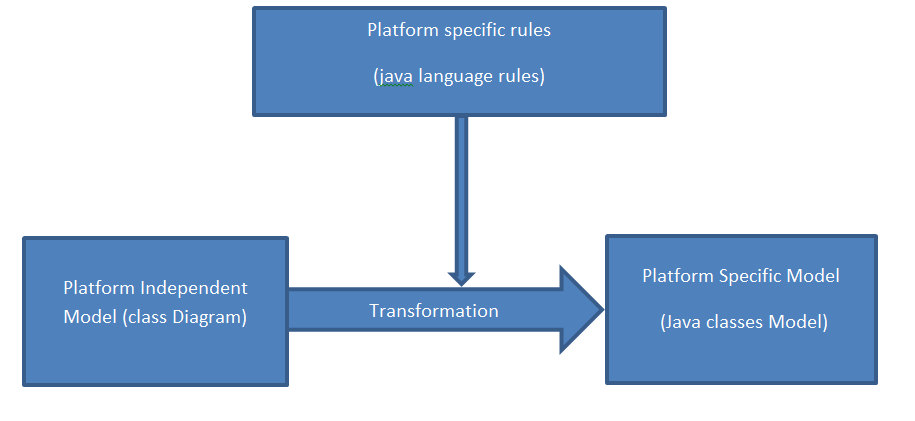


Figure 2 MDA Methodology

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 focusing on the requirements for the system, the technical implementation details being hidden.

### 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 relative to the availability of traditional developers (Ambler)

## Behavior Driven Design (BDD)

### 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.

### What is BDD?

#### 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.

#### Methodology

Applying BDD to the design of a 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 6.

As illustrated by the step 2 of Figure 6, once the features have been elicited, they are converted to plain text natural language description as user stories and scenario templatesusing a specific format: The Gherkins language as illustrated in Figure 3.

This is defined as follows:

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)

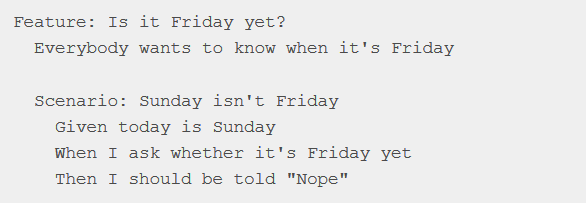


Figure 3: A feature “Is it Friday yet” with steps defined in Gherkins

After the previous step – as illustrated in step 3 of Figure 6, 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 3 - 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.

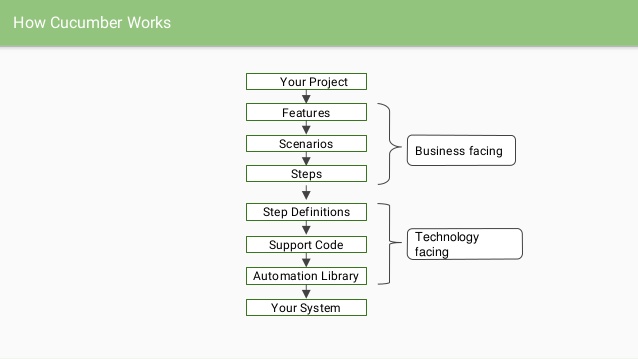


Figure 4: How Cucumber Works **[[2]](#footnote-2)**

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 5 where is defined the test code for the scenario defined in Figure 3. 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.

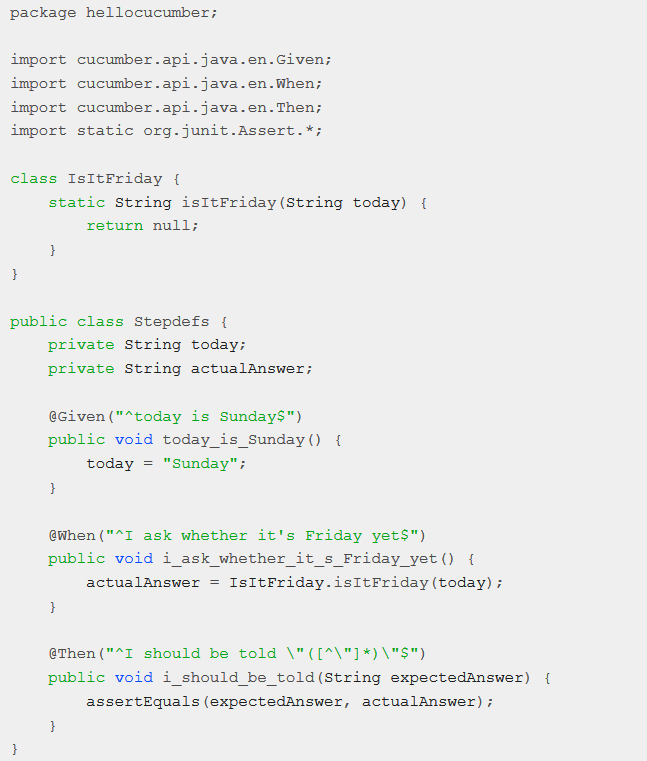


Figure 5: “Is it Friday yet” corresponding steps definition in a java test file

Finally, as shown in steps 4 and 5 of Figure 6 , 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.

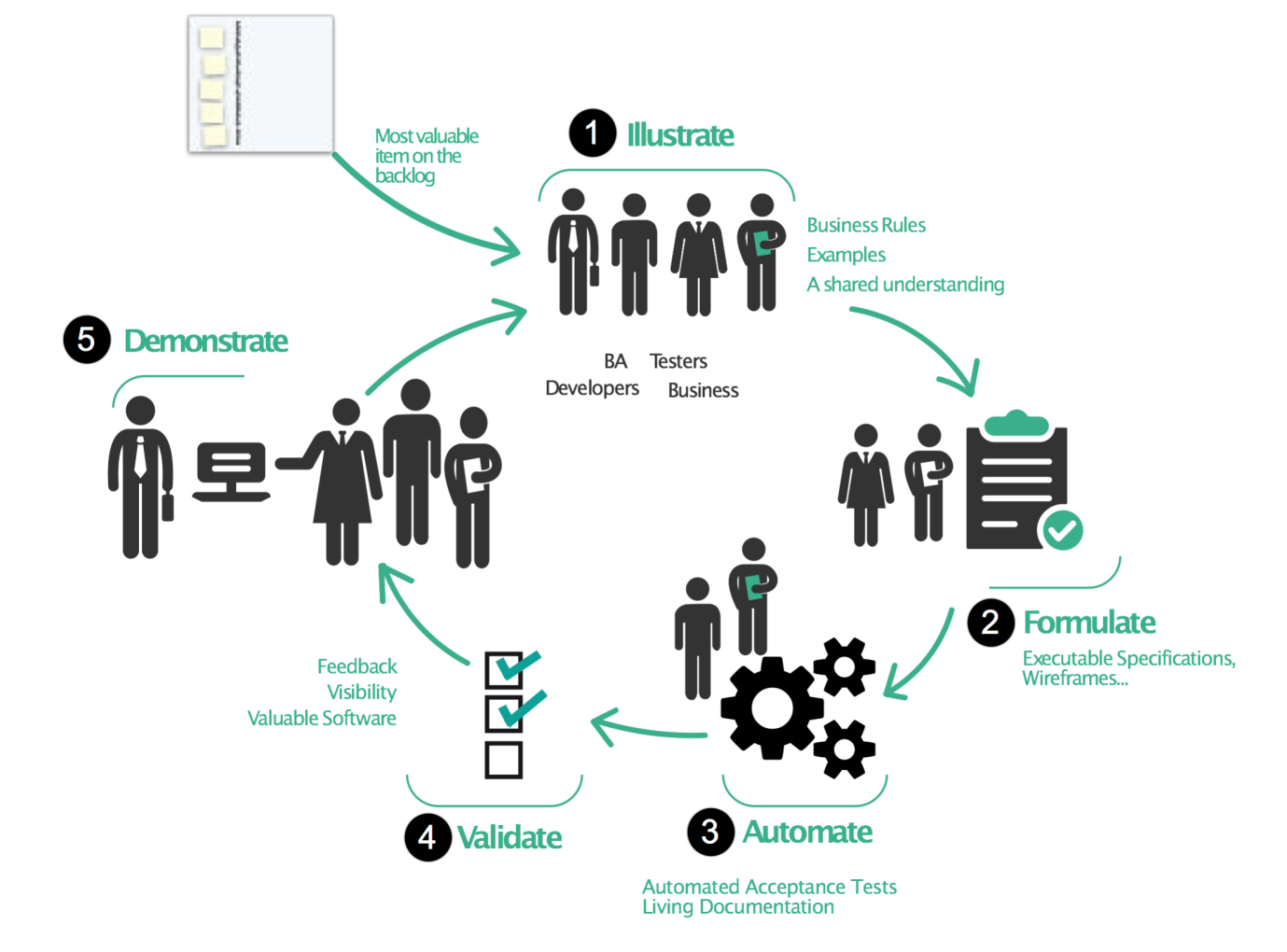


Figure 6: Behavior Driven Development methodology [[3]](#footnote-3)

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 process has 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.

### Drawbacks of BDD

BDD has some drawbacks that should not be neglected as they can impact negatively 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.

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 isn’t a 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.

## Domain Driven Design (DDD)

### History

Many softwares are made following a monolithic data centric architecture as illustrated in Figure 7 . 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.

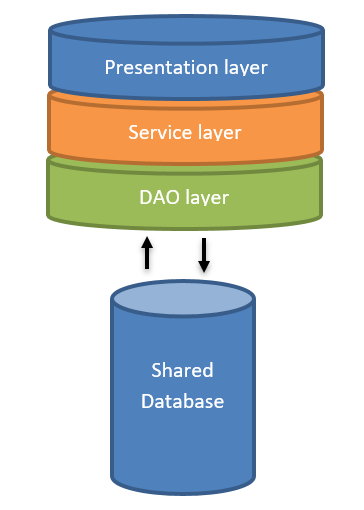


Figure 7 Monolithic Architecture

Such architecture, in complex 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)***

### What is DDD?

#### 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.

#### 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. Techniques used are Event storming, Agile workshops, Impact mapping, User storytelling, Etc. Interaction between developers and business guys is highly needed here.

##### 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 8. 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)***

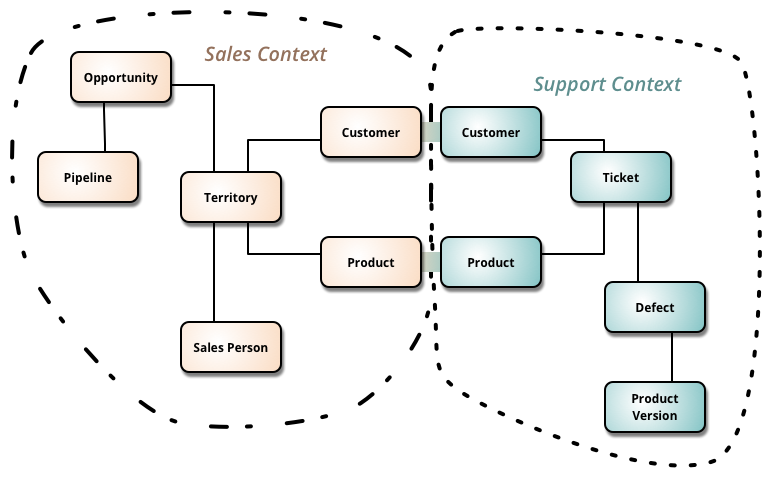


Figure 8: Two bounded Contexts independent from each other[[4]](#footnote-4)

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, 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.

##### 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 managing 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 9.

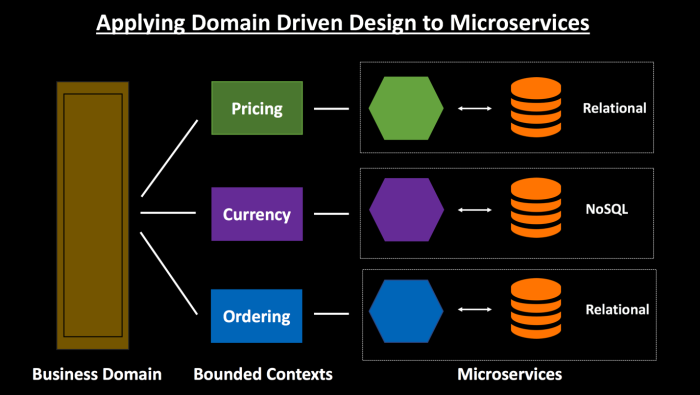


Figure 9 : One business bounded context, one micro-service, one domain model, one database [[5]](#footnote-5)

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, Dependency injection Principle…This paper will focus on an implementation with a hexagonal architecture.

A hexagonal Architecture, as illustrated in Figure 10 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 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 …)

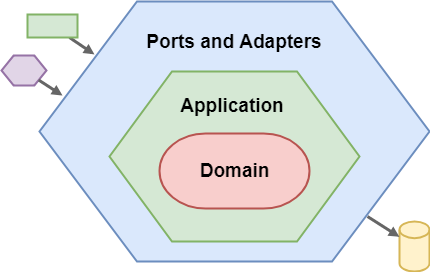


Figure 10: Overview of Hexagonal architecture[[6]](#footnote-6)

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…).

Around the domain layer, resides the application layer, within which domain services are injected in order to be orchestrated and serve to upper layers.

Others layers are above the application layer, they are 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, 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 11.

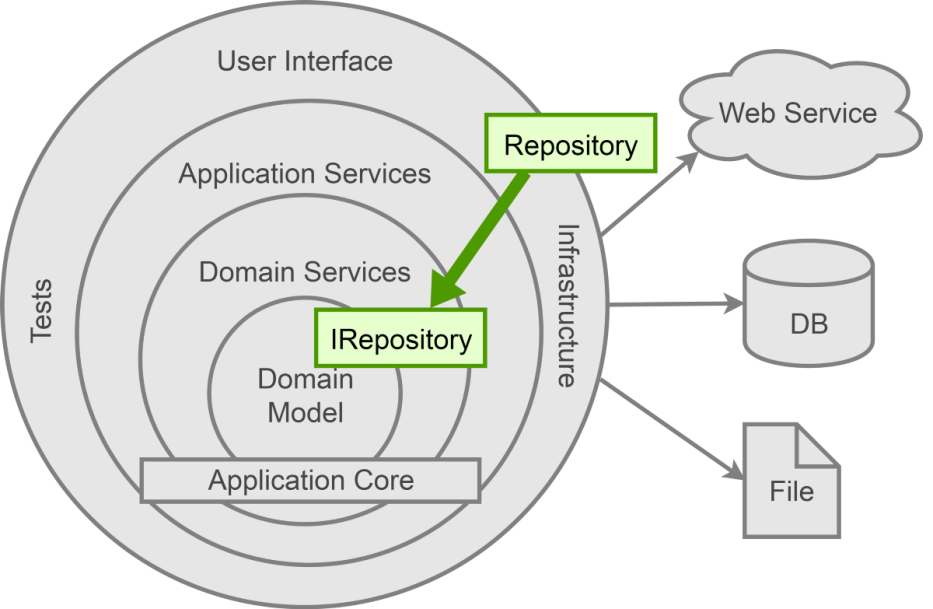


Figure 11: Extended Hexagonal Architecture: The Onion Architecture[[7]](#footnote-7)

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 doesn’t only focus on the knowledge of the subject (Collaborative Design), but give us useful ways (Strategic Design) of understanding the subject matter, as well as implementing maintainable and scalable software (tactical designs).

### Drawbacks of DDD

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

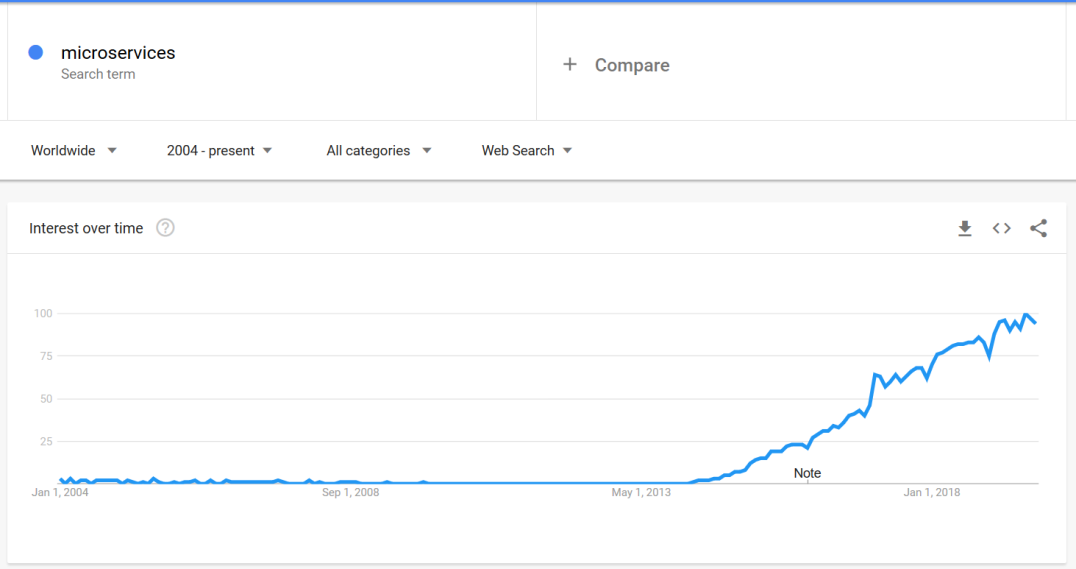


Figure 12: Micro-services searches trend since 2004[[8]](#footnote-8)

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’s 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.

## Conclusion

On our way to find means to tackle the problem of improving the of designing complex softwares, 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.

# Research Methodology: Qualitative analysis

## 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 doesn’t 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 been used in order to confirm hypothesis, but to discover new insights that are unknown as well.

### 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 theme 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 complex 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, was 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.

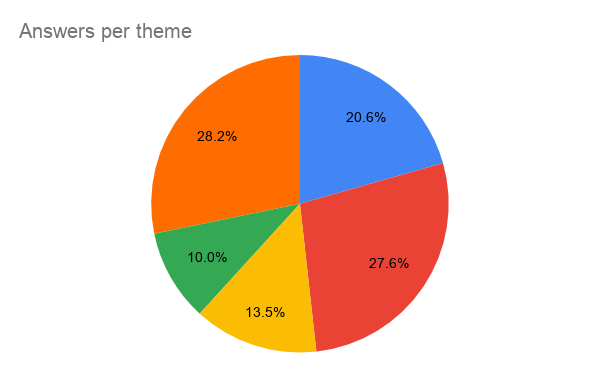
### Organization of the result grid (Themes and sub-themes)

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.

## 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.



**Knowledge on approaches**

**Satisfaction feedback**

**Usage of approaches**

**Issues and expectations**

**Stakeholders Vs. Roles**

Figure 13: What interviewed where likely to talk about

Key takeaways 1:

As Figure 1 shows, 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.

To discover other takeaways, we will analyze theme by theme.

### 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.

Key takeaways 2:

The responsibilities are not quite defined, not quite known nor applied

### 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%.

Key takeaways 3:

The standards approaches are not as used as it seems as every team/company/entity is applying it adapted to their needs. However the DDD is attracting more and more interest.

### 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.

Key takeaways 4:

Every stakeholder is feeling or has been involved in the overall software designing process. MDA is quite unknown, BDD is much more used than DDD but the DDD is recognize as a necessity but very few have the courage to get to it.

### Approaches satisfaction feedback:

Very few said here, the respondents where 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.

Key takeaways 5:

The profiles do not practice knowledge management about previous experiences with previous approaches. They do not gather what went wrong, what to do to avoid, what went right nor how to enhance. Besides, they feel that BDD and DDD can be used to deliver a better quality with respect to the delivered value and the maintainability.

### 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.

Key takeaways 5:

There must be a better interaction between business team and developer teams and a clear separation of responsibility in the whole chain.

## Results and comments

From the key takeaways we can extract the following information:

### The Organization refactoring

Many issues are raised for organizations are still running on SILO mode, that 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.

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 everyday as being part of the same team, the same product, the same spirit as illustrated in the new model of IT in Figure 14.

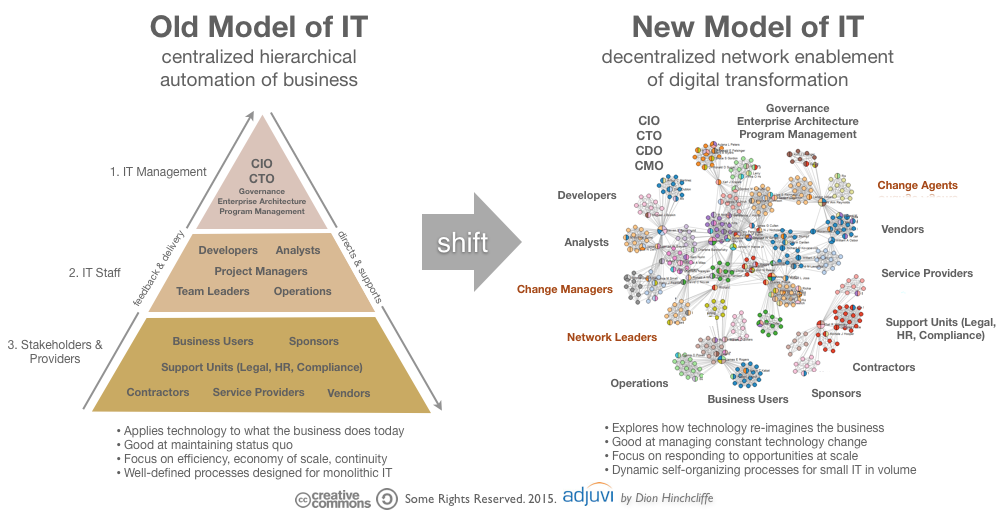


Figure 14: From pyramidal architecture to decentralized architecture[[9]](#footnote-9)

### The knowledge Management

The interview results pointed out that the profiles didn’t get enough data on past experiences; nevertheless, they did have some experiences on the evoked design approaches. So, they didn’t take advantage of their past experiences by creating, updating and controlling documentation about what went wrong? Why? How to correct? And how to avoid? To avoid the same errors to keep repeating, a knowledge management system, more specifically the living documentation must be set up.

### Demystifying the Domain Driven Design

Pretty much people acknowledge the necessity of using the Domain Driven Design, statistics agree with them but the DDD is not well known already even though it has been described since 2004. Actually, is true that DDD with its tactical patterns is bringing a new way to work with the business (provided that the organization is refactored to atomic mode), nevertheless, it relies on a bunch of design approaches (including Model driven design and BDD) and architectural styles that are already applied by many teams all around the world. So there must be an evangelization around it in order to demystify it.

Now that we’ve got some insights about the issues, causes, we will aggregate (Table 3) that with what we already have from the state of the art in order to have a better visibility of how to enhance the making of complex softwares within organizations.

Table 3 Suggested solutions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Research problem | Fine Grained Issues | New insights | | Existing solutions |
| DDD+BDD+MDA |
| **How can companies improve their way of designing and Developing complex Software?** | **How to educate the teams and let them notice the importance and the priority of getting aligned with the business?** | Break the silos  From project mode to product mode  The product mode enforces the decentralization of the architecture of the company from silo mode to architecture across the business, for the business and by the business. | Manage knowledge and feedbacks  Living documentation | Educate, train and form around DDD as it ships within its patterns BDD and MDA and well known architectures such as the micro-service architecture. |
| **How to speak to the business people and to captivate their interest?** | **Collaborative Design:**  Event Storming/knowledge crunching  Create the Ubiquitous Language |
| **How to get the most correct domain knowledge as possible?** |
| **Which development pattern helps the team to focus on domain issues as described by domain experts?** | **Strategic Design:**  Detect Bounded contexts  Edit platform Independent Models  Edit automated behavioral Tests |
| **How to separate technical implementation concerns from domain logic issues?** |  |  | **Tactical Design:**  Hexagonal & Onion architecture  +  Single responsibility principle  +  Dependency injection principle |
| **How to apply an architectural style facilitating scalability and features enhancements?** |

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

## Organization refactoring within AXA Bank ISD

### Overview

As an apprentice within the ISD, 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’s 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, comprising:

* Lean-Agile Leadership: Lean-Agile leaders drive and sustain organizational change and operational excellence by empowering individuals and teams to reach their greatest potential,
* Team and Technical Agility: Lean-Agile principles and practices Agile teams need to be high-performing and create high-quality, well-designed technical solutions that support current and future business needs,
* DevOps and Release on Demand: Principles and practices of DevOps 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 inform 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 operations, and Lean governance for a SAFe portfolio. (SCALED AGILE ,Inc., 2018)

The whole transformation process is beyond the scope of this paper, we will focus on how the Team and Technical Agility competency have been implemented. Moreover, it is an effective way to assess some of the suggested solutions to our research problem. Since it relies, besides others on:

* The decentralization of decision making: on one hand, equipping teams with the domain experts; on the other hand, equipping individuals with judgment needed to make good decisions: breaking the silos.
* The application of system thinking: this is about everyone committing and thinking to the common goals of the larger system: being committed within the product mode.

### Team and Technical Agility transformation

The implementation of this competency is done around a Program increment (PI) - a timebox 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.)

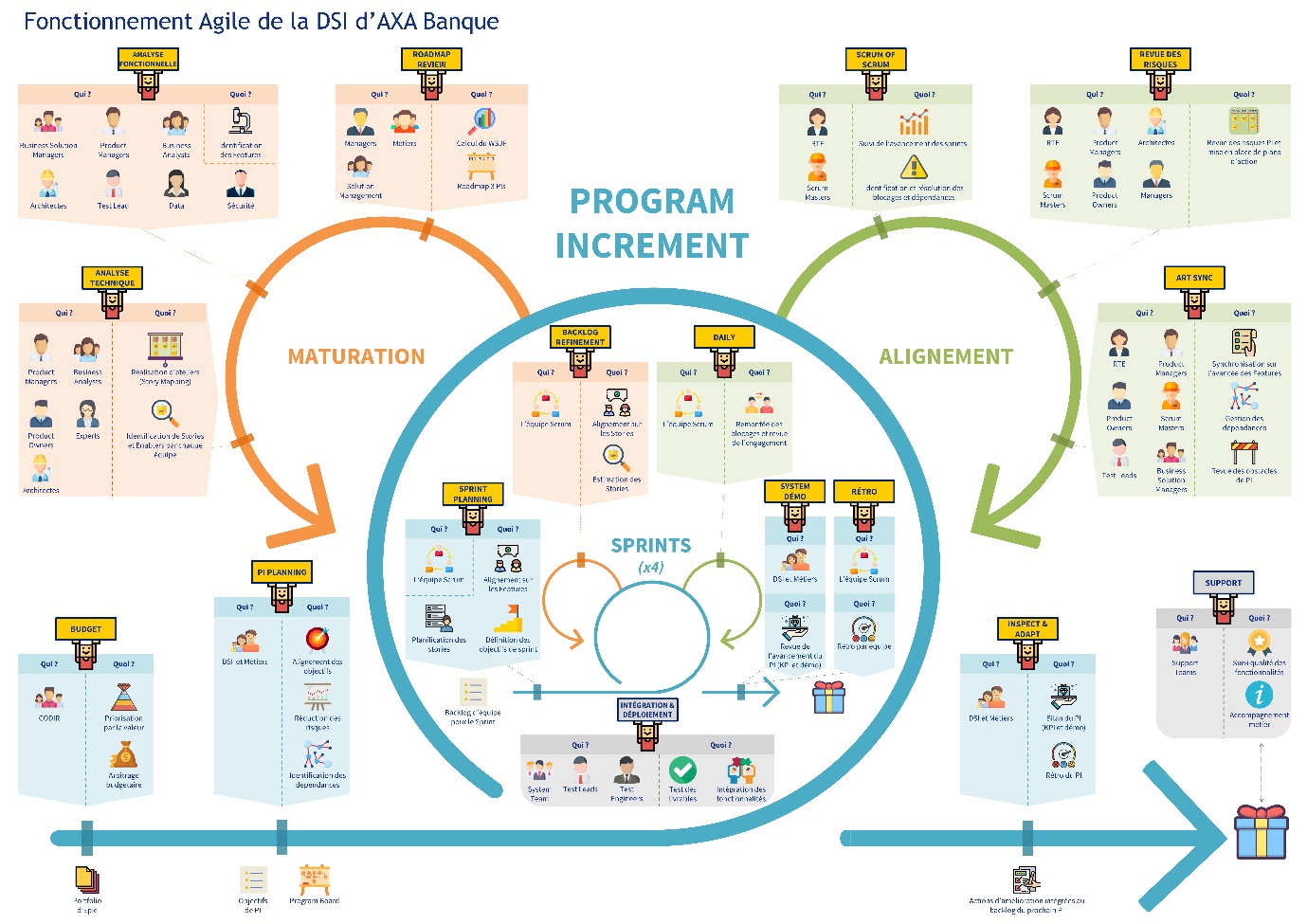


Figure 15: Team and technical Agility Process of AXA Bank ISD

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

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

#### Program Increment

It starts by the definition of its the 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 Sprint of 2 weeks each.

A sprint is about:

* Spring planning: Definition of sprint objectives and backlog - the set of items to be developed by the Scrum team,
* Daily meetings: Giving daily updates about what is being developed and achieved,
* Integration and deployment: continuous integration and delivery of the new features to the previous delivered ones,
* Backlog refinement: Making sure developments are aligned with what was supposed to be built,
* System demo and retro: Presenting what have been developed during the sprint and feedbacks on what went eventually wrong, how to enhance it or avoid.

#### Alignment

As its name highlights, here we make we are still on the same track with what have been defined during the PI planning, if not, raise the issues and unblock them as fast as possible, so that the train can continue to go forward. The alignment operation consists of 3 operations, namely:

* SRUM of SCRUM: Done by the Release Train Engineer (RTE) – the one in charge and coaching the ART—with the Scrum Masters, during which, the ongoing spring are monitored, and potential problems are raised and resolved.
* Risk review: Here, the PI risks are analyzed, and actions are set up accordingly. This is done with the collaboration of the RTE, product managers, architects, scrum masters, product Owners and managers.
* ART synchronization: Here, the RTE, the Product Managers, the product owners the Scrum masters, the test Leads, the business solution managers meet in order to get synchronized on how features are getting developed and treat potential dependency problems between features.

#### Maturation

This stage happens just before the beginning of a new PI. The aim is to:

* Do a Roadmap review of the 3 previous PIs including the current one,
* Do a functional analysis of what can be done for the next PI. This is where the feature discovery happens with the helps of Business Solution Managers, product managers, Business Analysts, Architects, test lead, data engineers and the Security guys.
* Do a technical analysis: Workshops for identifying new stories and enablers for every team.

At the end of every 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.

When the product is delivered after a PI to the end customer, the Support operation begins thanks to the support teams.

### Key Performance Indicators (KPI) and appraisal

In order to assess this transformation across the different processes of the ISD, we must first highlight the KPI that are:

* Released features frequency
* Commitment achievement rate
* Released Stories frequency
* Released story points
* Features ready for the next PI
* Number of Incident reported
* Number of Bugs reported

The first PI occurred in January, since, three PIs have been achieved, over 87 epics that are either done or in progress.

The following table shows clearly the delivery enhancements over the PIs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Before | PI1 | PI2 | PI3 |
| Released features |  | 64 | 120/238 |  |
| RESPECT FOR THE COMMITMENT |  | NA | 57% |  |
| STORIES DELIVERED |  | 964 | 1160 |  |
| RELEASED STORY POINTS |  | 2550 | 2898 |  |
| Features ready for the next pi |  | 40 | 86 |  |
| incident reported RATE |  | 13% | 18% |  |
| Number of bugs reported |  |  |  |  |

## Demystifying the Domain Driven Design

As previously mentioned, an evangelization around DDD was necessary in order to demystify it. The bank decided to start that process thanks to an architect practicing DDD since many years. The process of evangelization consisted of

# Conclusion and further work

# Table of acronyms

BBOM: Big Ball Of Mood

BC: Bounded Context

BDD: Behavior Driven Design

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

QVT: Query/View/Transformation

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)

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.

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/

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

1. Quelles pratiques/méthodes/approches de conception/développement logiciel connaissez vous? particulières durant le cycle de vie ?  
     
   *Si ils ne citent pas l’un de ces éléments: MDA,BDD,DDD* ***passer au 5, sinon passer au 6***
2. Avez-vous entendu parler des approches de conception logicielles suivantes:

MDA  
BDD  
DDD ?

1. En quoi consiste généralement les méthodes MDA,BDD,DDD ?

Sous thème 3 : Utilisation des approches de conception logicielle

1. Avez-vous déjà appliqué(été impliqué dans) une de ces ces approches lors de vos précédentes interventions ?
2. Qu’en est-il de votre intervention actuelle ? êtes-vous impliquez? L’appliquez-vous ?  
   *Si non poser la 9*
3. Pourquoi ne l’appliquez-vous plus ?

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

1. 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 développeurs** :   
   Améliore t-il la collaboration avec les PO et BA?  
   Améliore t-il la qualité, lisibilité, du code, réduisent t-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?
2. Selon vous, les pratiques actuelles sont-elles évidentes,pas assez évidentes ou difficiles à mettre en oeuvre ? / les façons de procéder actuellement vous conviennent elles ?

Sous thème 5: Attente concernant les approches de conception logicielle

1. 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?

## Questionnaire for the quantitative analysis

(join the pdf when printing)

1. Source : https://www.omg.org/mda/ [↑](#footnote-ref-1)
2. Source : https://www.slideshare.net/kalhanrl/cucumber-for-automated-acceptance-testingpptx [↑](#footnote-ref-2)
3. Source : https://johnfergusonsmart.com/so-you-say-you-are-doing-bdd-the-story-of-the-whiteboard-and-the-nail-gun/ [↑](#footnote-ref-3)
4. Source : https://martinfowler.com/bliki/BoundedContext.html [↑](#footnote-ref-4)
5. Source : https://dzone.com/articles/breaking-the-monolithic-database-in-your-microserv [↑](#footnote-ref-5)
6. Source: https://github.com/cwoodruff/ChinookASPNETCoreAPIHexArch [↑](#footnote-ref-6)
7. Source : https://dzone.com/articles/stem-in-onion-architecture-or-fallacy-of-datanbspl [↑](#footnote-ref-7)
8. Source : https://trends.google.com/trends/explore?date=all&q=microservices [↑](#footnote-ref-8)
9. Source : https://dionhinchcliffe.files.wordpress.com/2015/05/old\_it\_versus\_new\_it\_networks\_of\_change\_agents\_enablement.png [↑](#footnote-ref-9)