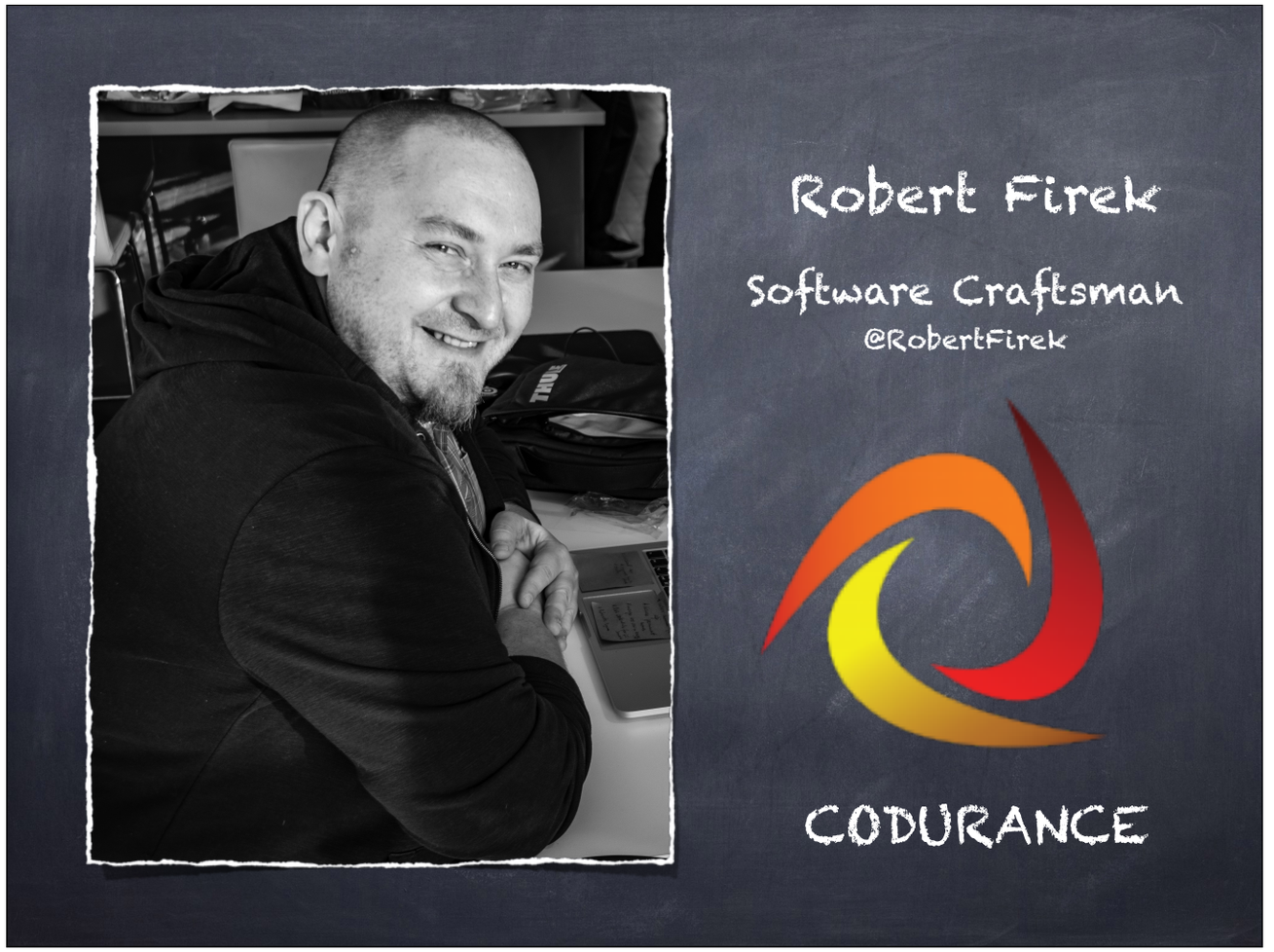
Agile, Craftsmanship, DevOps and other monsters

How to fight with them.



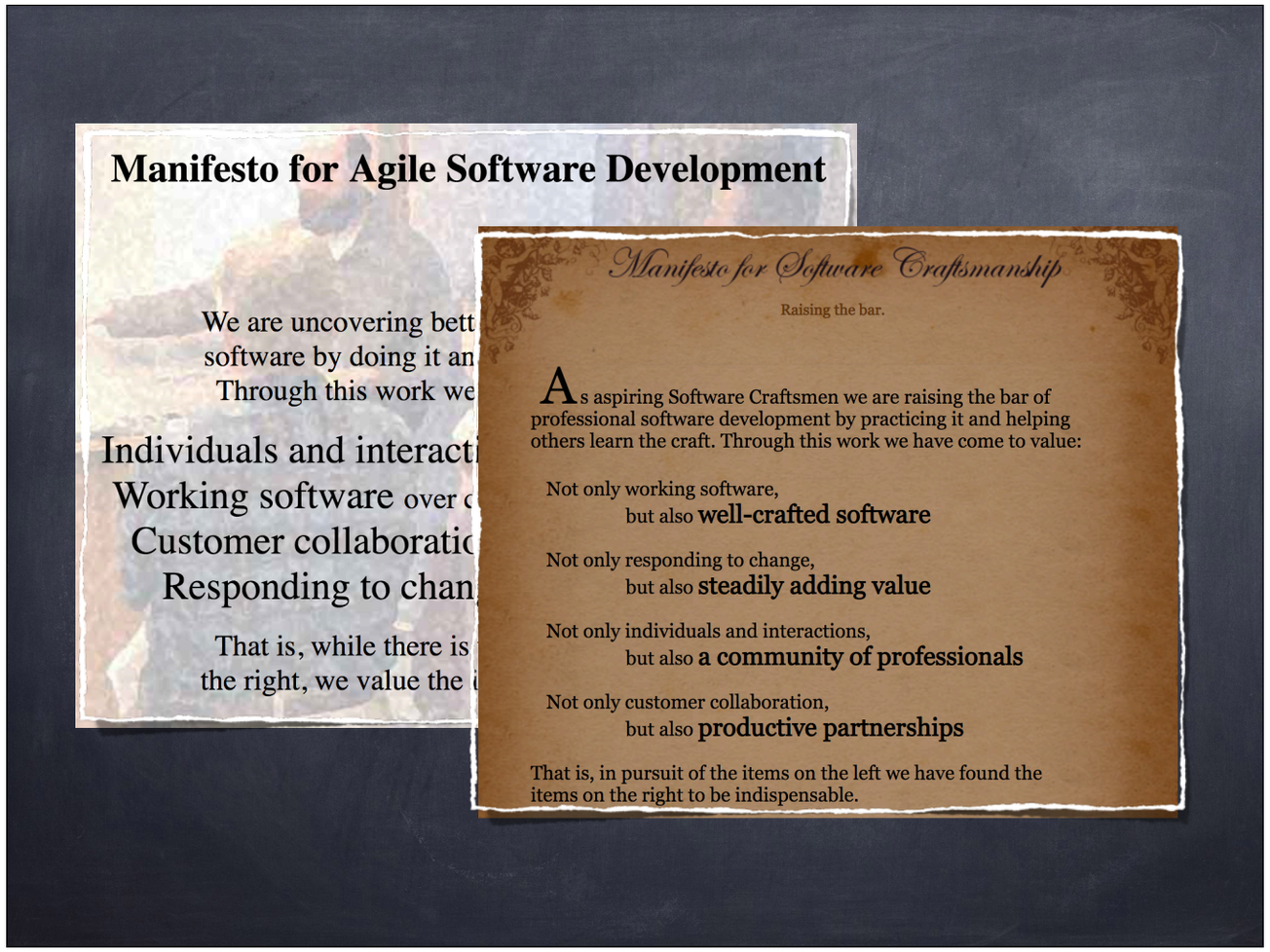
I’m the software craftsperson at Codurance. The software consultancy company where **we deliver value by crafting exceptional software.**

What we do in Codurance is related to the topic of my today's talk. Our statement looks like the excerpt/paraphrase of one of the points from the software craftsmanship manifesto. We are **advocates** of Agile, Craftsmanship and Devops and any **pragmatic approach which help us to deliver value to our customers.**

Unfortunately, being a practitioner is not an easy job. Many times, we encounter a lot of resistance from people. “It doesn’t work for us.”, “We already tried this approach and it failed.”, “Our company is different.” are statements which we can hear during my work.

In my opinion, the source of this resistance has one main reason (among all others). It is a misunderstanding of practices which they introduce in their project.

Let’s take a closer look at all these practices and why they may not work in all places and what kind of mistakes we do.



As I said, in Codurance try to **share with our customers** the values of two manifestos.

It is **Agile and Software Craftsmanship Manifestos**. I can say that it is **not an easy task** to share these values because the understanding of these documents can be different in different places. Different people have different opinions about what is Agile and Software Craftsmanship.

I don’t blame people to have a **diverse view** on Agile, because if we look at the Agile manifesto…

We can find **only a few sentences** which summarise the whole software development process.
All years of exploration, development, failures condensed in a few sentences.

Is in it amazing? We have all the answers in this script and all we have to do is to value:

- **Individual and Interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

How many of us actually try to introduce these rules in the real live?
Do you really value all these things over relicts of Waterfall?

My experience saying that we are not always align with these values. On the market, it is hard to find a company which is not “agile” but it is hard to find a company where all people involve in Agile process read and understand the manifesto.

Do you know that the Agile manifesto defines also…

Twelve Principles behind the Agile Manifesto.

Who read these principles? [raise your hands…] I didn’t read it when I read the first time the manifesto. I didn’t need to read about:

- customer satisfaction
- embracing constantly changing requirements
- delivering working software frequently
- business people and developers working together

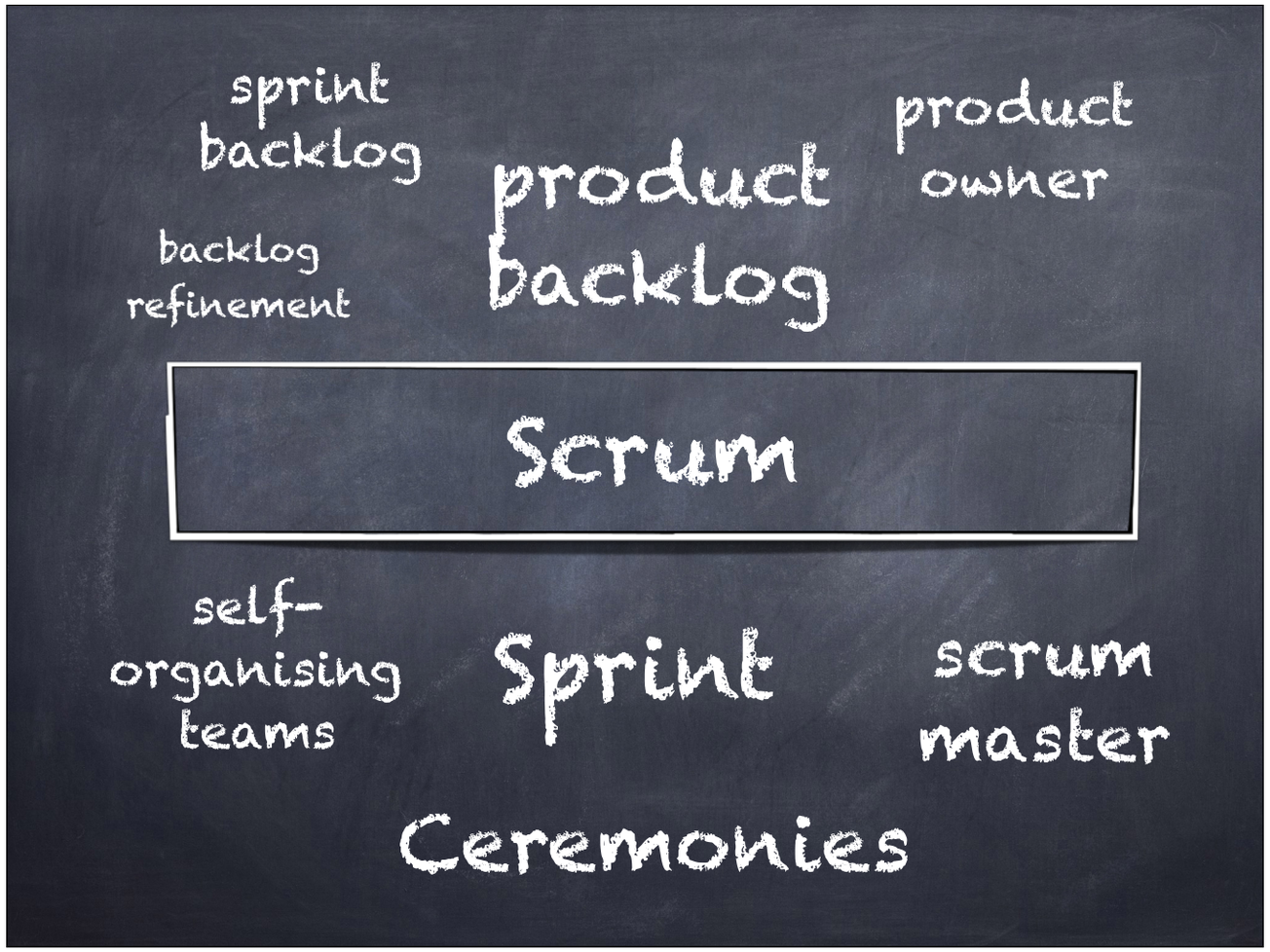
I didn't need to read about:
- building projects around motivated individuals
- working software as the primary measure of progress
- self-organizing teams

And I didn't need to read about:
- simplicity

and few other things included in these twelve simple principles.

You may ask why I didn't need to do that?

The answer is simple. I had a solution for all that problems ….



…big, fat Scrum gave me all the answers.

We focused on the **customer satisfaction by defining backlogs**. We listened about the needs of our customers during **the backlog refinement**. We defined the stories, estimate and write acceptance criteria.

We finally define the sprint and implement stories in self-organised teams.

And we had the master of ceremonies - the scrum master who keep the process running smoothly.

This was my replacement for 12 Principles.

That is the ideal view of Scrum was my first interaction with Agile. During my 8 (maybe 10) years of experience with Agile and Scrum, I've seen only once a good implementation of Scrum framework. **Only once** I collaborated very close with the business to understand their needs. **Only once** I defined stories which has really deep meaning from the business perspective.

Only once I applied almost all twelve principles in such a way that developers and business people were happy.

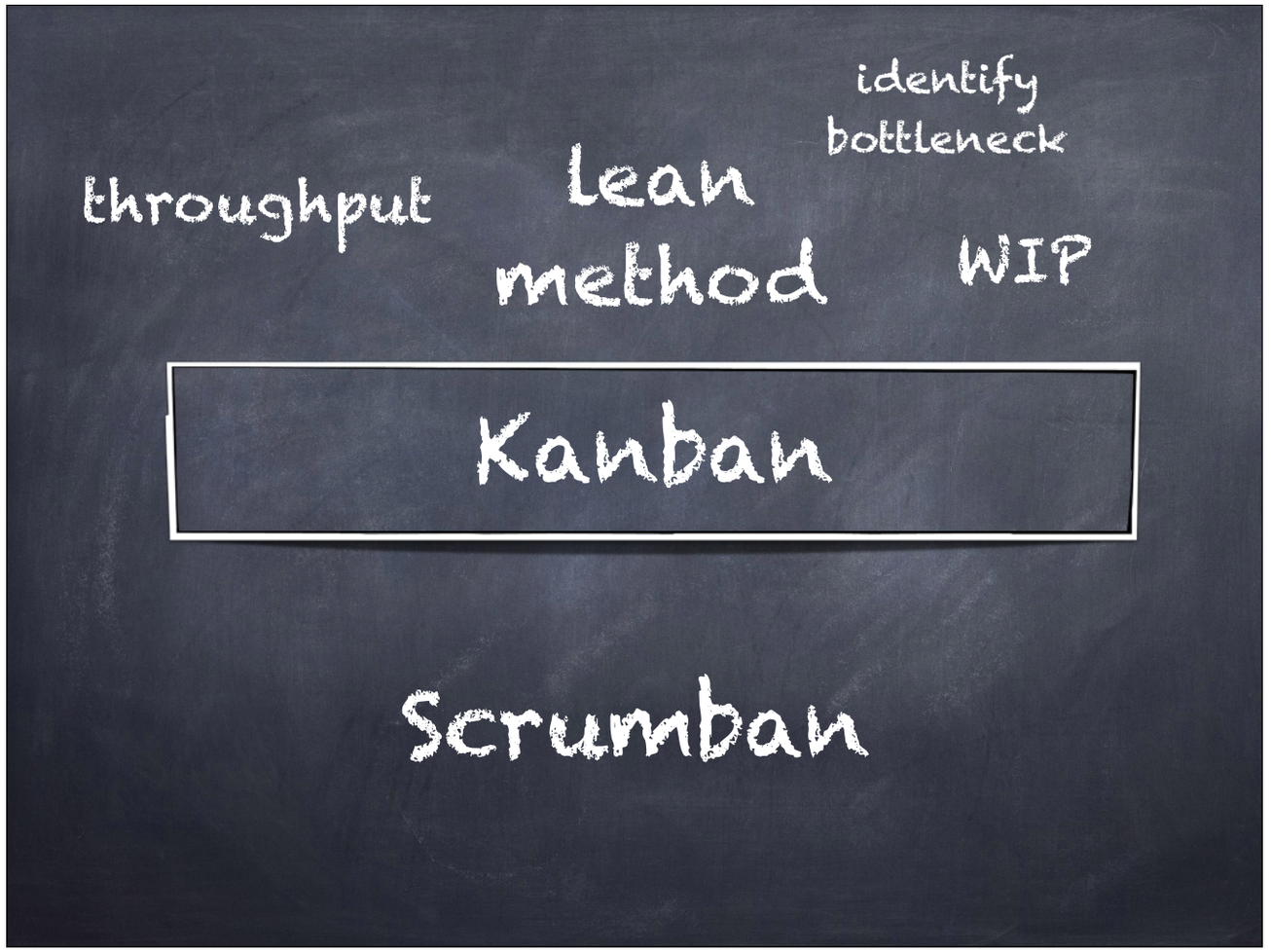
The Scrum is **the biggest advantage** and **the biggest weakness** of Agile movement.

It is the biggest advantage because it allows us all smoothly move from Waterfall process to the approach where we are more align with Agile manifesto principles. By using well-defined building blocks (like backlogs, ceremonies) we can drive our customer to change they thinking about software. To make them Agile.

Scrum is also the biggest weakness because it allows people to stay where they were. It allows to rename Waterfall steps and continue to work in Waterfall style. Backlog is our contract which we cannot be change. Sprints became milestones and Scrum Master is our former Project Manager. We just change the names and run everything as usual.

I'm not saying that Scrum is only bad. The bad part is in us. People. We have to change our behaviours because **being Agile doesn't mean to follow blindly the methodology like scrum. Being Agile means to be focused on the customer needs and business values by continuous improvements.**

Some may say that the answer for problems of Scrum is…



…Kanban.

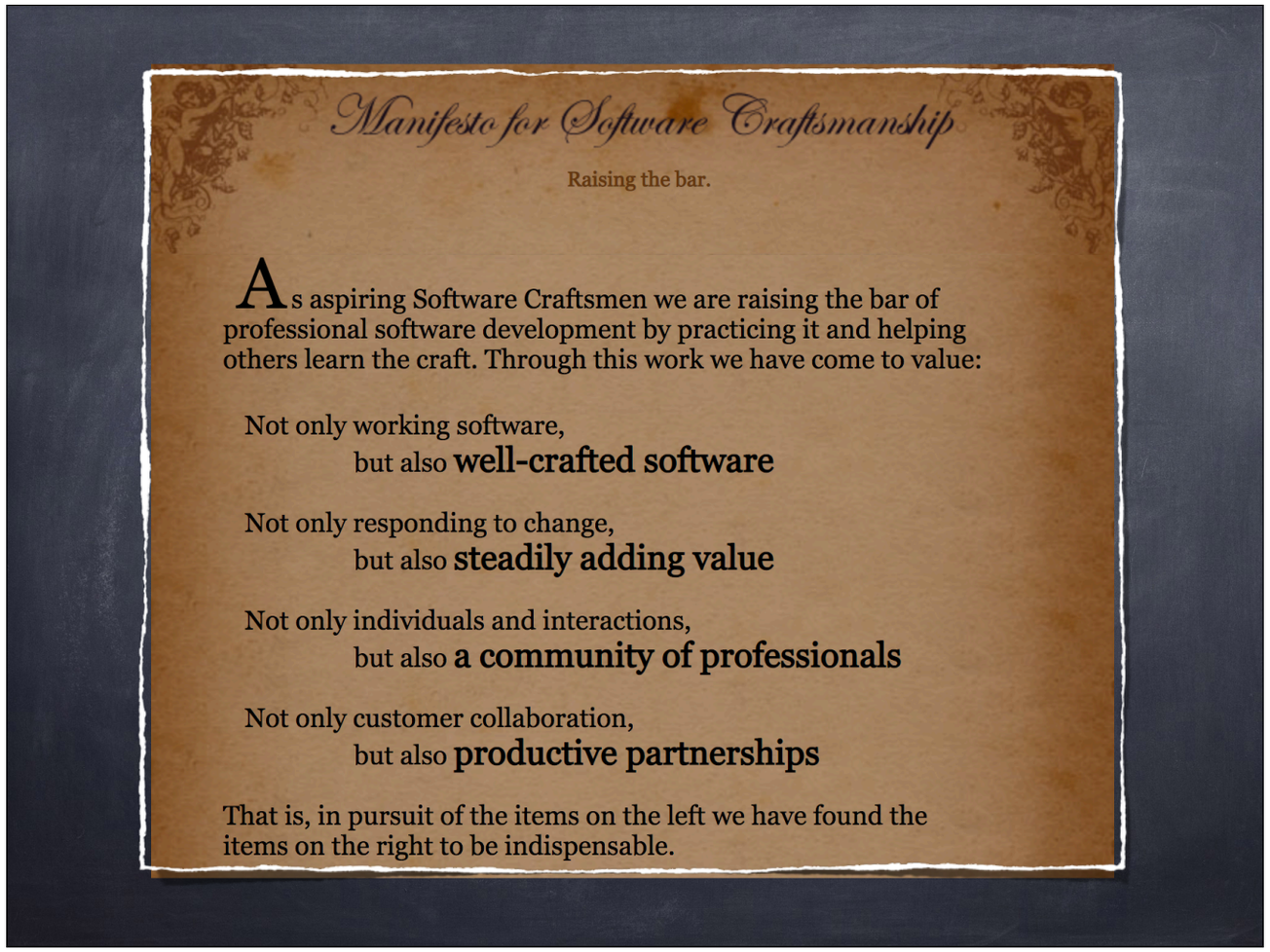
I think this is another misunderstanding. Kanban in Software exists because we started to learn from other industries. Kanban is the industrial method to improve the process by scheduling the process in the right way to avoid waste and bottlenecks. In Kanban, we identify issues and fix them as soon as possible to keep constant pace of our work. We also **use techniques like “Work In Progress” to keep throughput of our tasks trough our development process.**

The idea of Kanban is very close to Agile principles but it doesn’t solve problems which are emphasised in the Agile Manifesto because of Kanban’s narrow focus.

We should know what kind problems are solved by Kanban. It is important to understand that Kanban is not substitute Scrum.
We have another methodology - Scrumban, a combination of Scrum and Kanban, which complement Kanban. The cadence of work, some ceremonies and elements of collaboration are required to enrich Kanban and create fully functional methods which focus on Software Development.

[PAUSE]

This was very short introduction of our first monster. Let’s face the second one.



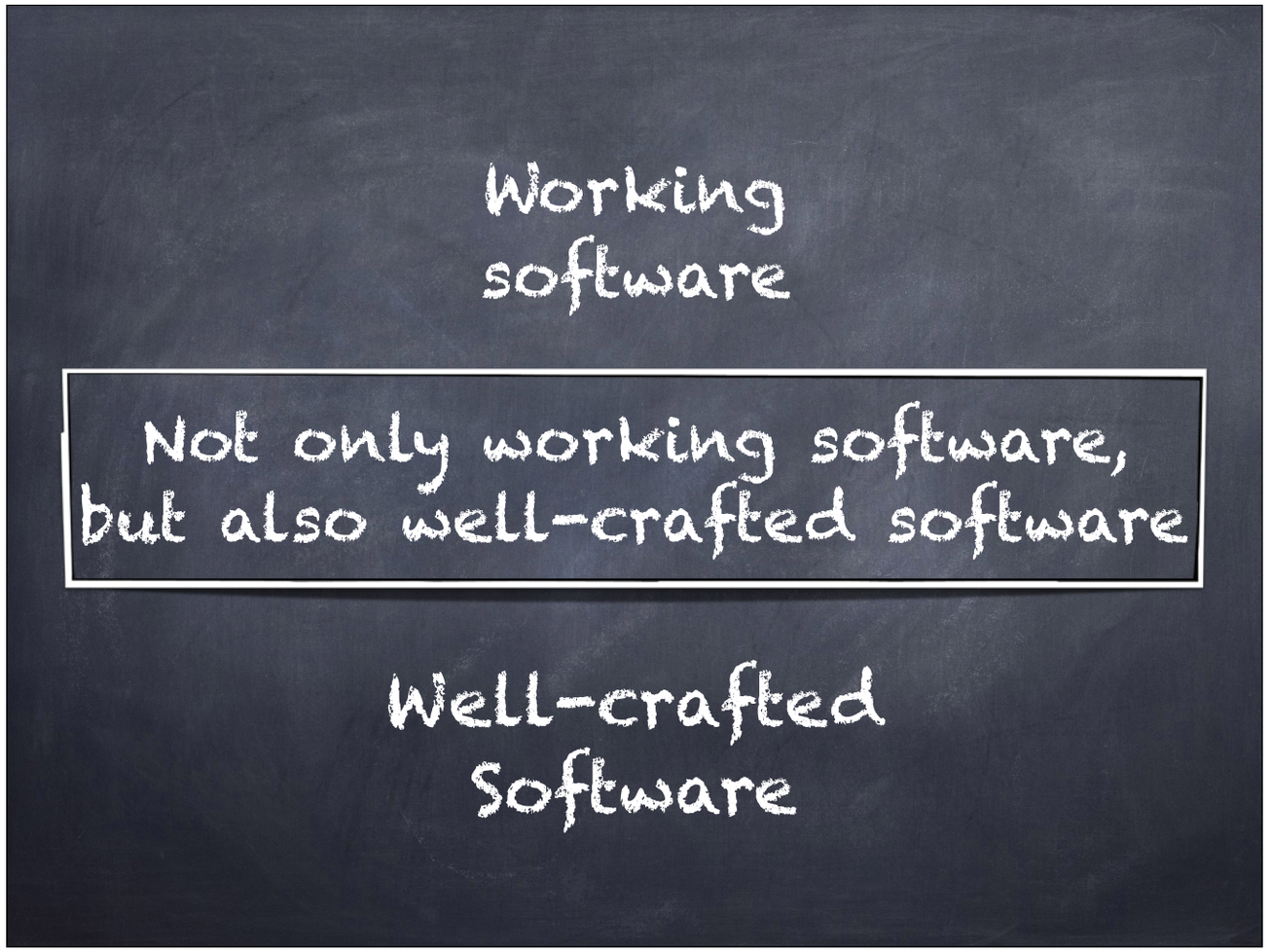
… the Software Craftsmanship manifesto.

The extension of the Agile manifesto. **The attempt to explore the technical side of Agile methodology.**

I have to be honest with you. I always had a problem with this manifesto because people around this movement never create any framework like Scrum for the Software Craftsmanship. Before I joined Codurance, I didn't even hear about this manifesto.

My initial problems with this manifesto was also connected **the vagueness of definitions** given by this document.

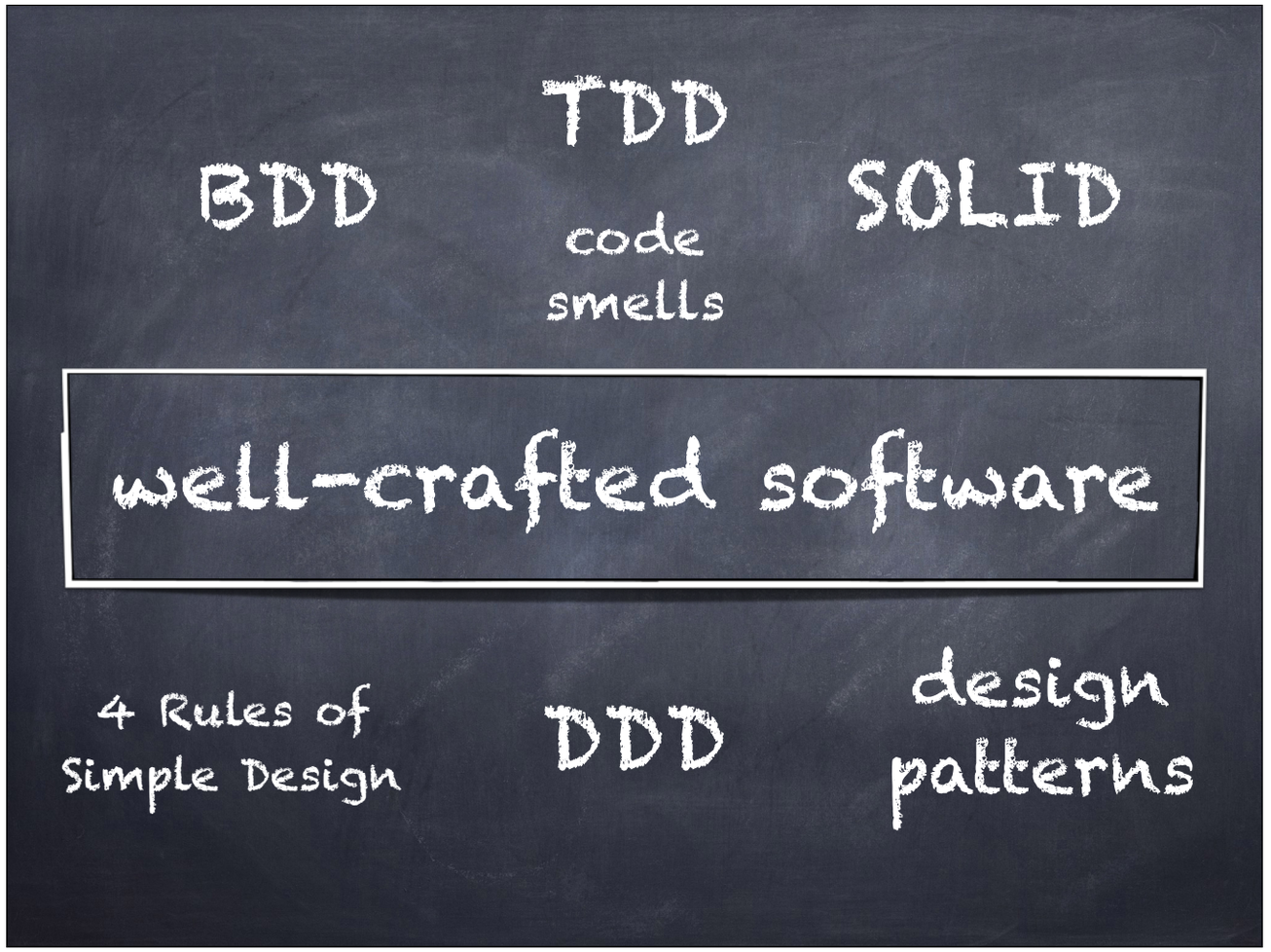
At the same time, **this lack of the precisions** forced me to think about **the meaning and the influence of each sentence on my work.**



The first value of SCM is about the well-crafted software. When I read it first time I was confused. I could see obvious relationship with the Agile manifesto but

What is the difference between working software and well-crafted software?
What does it mean to create a well-crafted software?

I started to think. In our industry, we have a lot different practices which improves different parts of our software. At the beginning of my carrier I didn't know too much about them. But later I've added to my toolset…



… different styles of Test-Driven Development.
I explored Behaviour-Driven Development.
I started to recognise Code Smells in my code.
I've learned and applied SOLID principles.
I read book about 4 Rules of Simple Design.
I finally understood the role of Design Patterns and the meaning of building block in Domain-Driven Design.

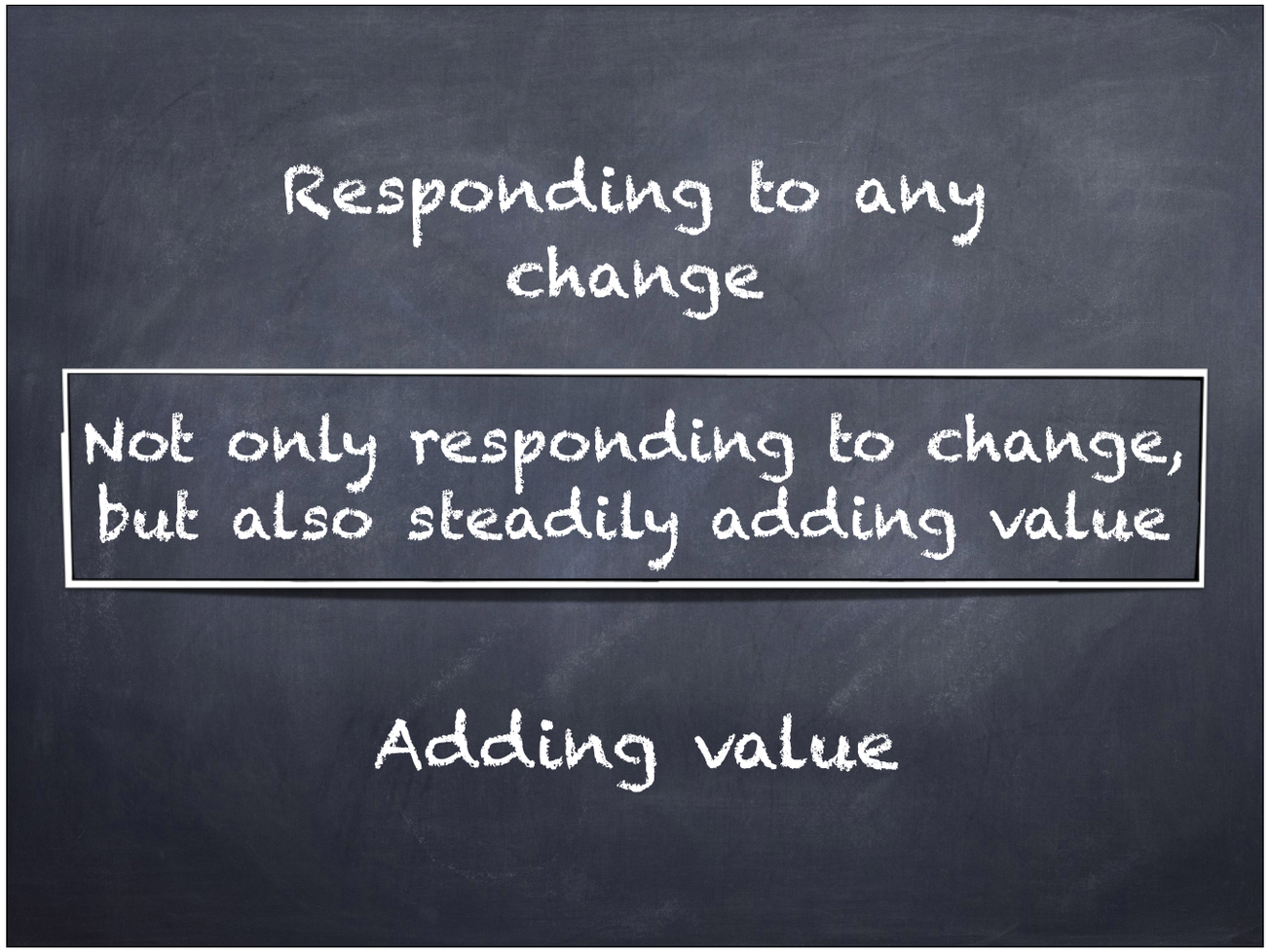
All these elements are the building block of well-crafting a code.

We should learn, understand and EVEN improve them. We should treat them as a necessary base for building any software. **This is my understanding of well-crafted software.**

When I try to introduce these building blocks, I always find someone who will say: **"It is not allowed to do these things in our organisation." or "It is waste of time.".**

I always have one analogy to change this attitude. **I called it the Medicine Analogy.** Imagine such a situation. You go to your doctor. Something is wrong with your body. **You would never like to be treated by a doctor who is not educated and who doesn't use the best possible methods to cure you**.
In the same way, your customer don't want to have crap software created by you. They pay a lot of money to you to create a good software.

This is very strong analogy (because in most cases we are not responsible for a life of other people) but in my opinion, **we should expect from ourselves the same quality as doctors expected from themselves.** We are not creating some support devices which helps other. **We are building the modern world. Software is everywhere.**



Good. This point will be easier.

I read this statement few times and I could not understand at the beginning what is all about. After a few readings I recalled all my projects and **it was epiphany**.

One of the biggest issues of Agile is the way we response to our customer. During my first projects in Scrum, we reacted on all changes requested by our business people. Bugs, change of requirements in the middle of the sprint, anything. We didn't care what it is.

We never reassessed the value of these changes to the project. We never asked if they **bring any business value** and, what is even more important, **how they will affect the delivery of currently implemented feature.** We blindly the followed the rule: "Responding to change over following a plan".

This behaviour created untracked work which was invisible to everyone and in consequence lead to conflicts because **we didn't delivery what we promised**. This point clearly tries to show us that our work should always be meaningful in the context of the business.

The next point of the Software Craftsmanship manifesto is about cooperation.

We met all here to increase our knowledge, to find out how we can improve our work, to help our customers having better experience. **We create a community of professionals who are happy to continuously develop their skills.**

This point force us to move outside our comfort zone and explore different possibilities of developing the software. Before I joined Codurance, I didn't have enough **courage** to give presentations or to help organising community events. **Now, the active participation in my communities is the part of me.**

In your case, it doesn't have to be the same path. It can any be activity which helps building groups of people who are happy to share and acquire knowledge about software development. You can participate in conference like this [thank you for being here], help to organise events and, what is the most important, always learn from others and teach others.

And the last point which I completely didn't understand when I read it the first time.

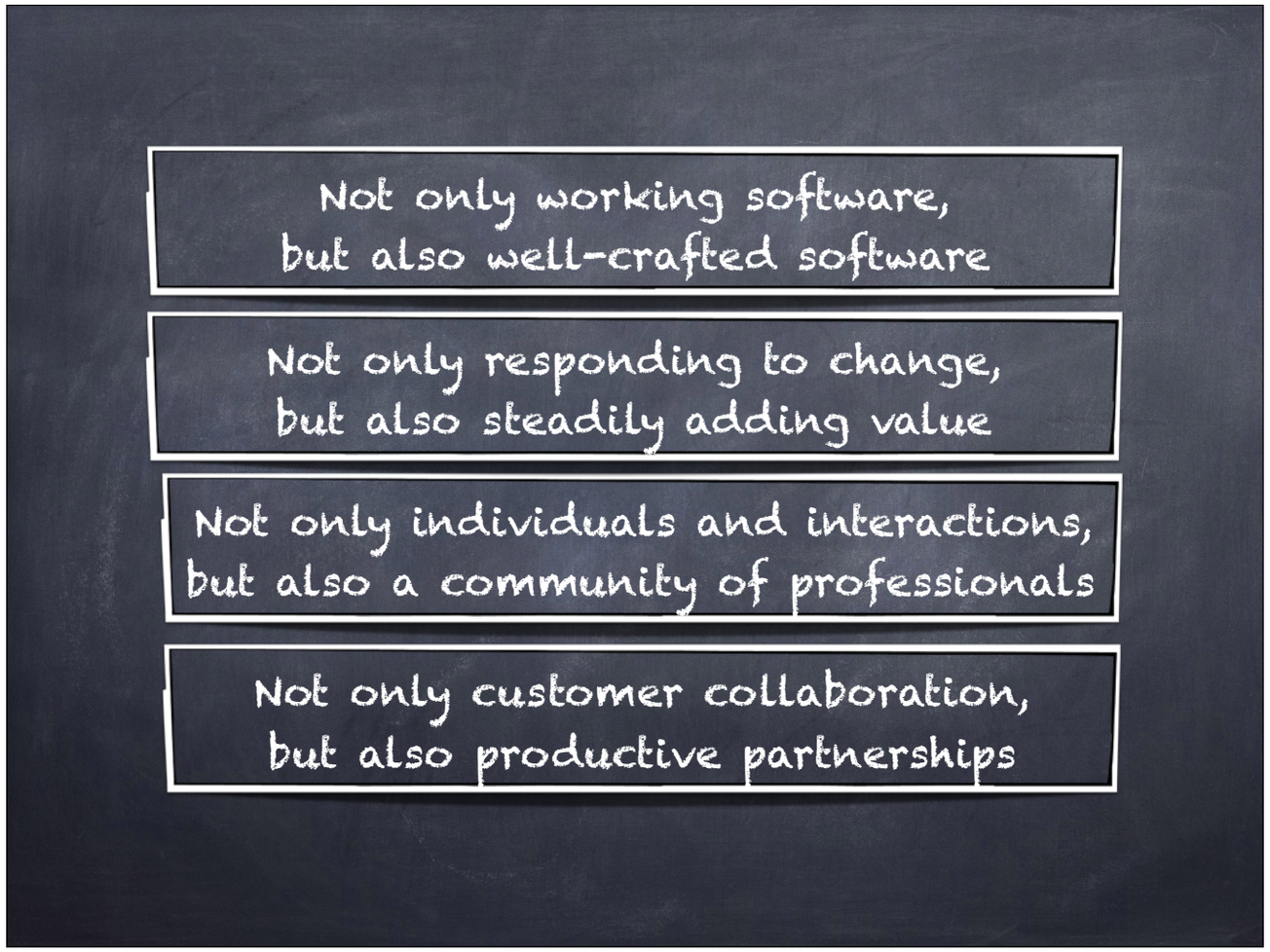
I could not understand, in my naiveness, why the customer collaboration may not be productive. We COLLABORATE so we should be productive.

A few projects later, I understood that the PARTNERSHIP part is about mutual respect of both sides. Plenty of times, in a good faith, you give a good advice to your customer but they gracefully ignore this advice. You try to do you best but they derail your efforts of bringing project to the next level. **This style is not a partnership, it is old way of thinking about software development as profession which is like assembly line in the factory where you receive tasks and execute them.**

We are not without sin in these situations. We are not alway good partners. First, sometime we focus to much on the technological side of the project. We have a nice ability to make technological choices based on our own, private development. We choose Kubernetes, Node.js or any other sexy technology or language without thinking about the impact of these choices on our productivity and the delivery of the software.

Second, we are playing really dirty in some cases and we exploit the lack of technical knowledge of our partners. We don't explained them in the **understandable, non-technical way our choices and the limitations of our tools.** We need to learn explaining the complexity of our problems to our business PARTNERS to allow them to understand that "adding a button to the web site" is not only work for our UI designer.

This is why it is so important to have mutual respect for each other's experience and knowledge and we have to build the trust that both sides have the same goal - **the best possible software which gives the business value.**



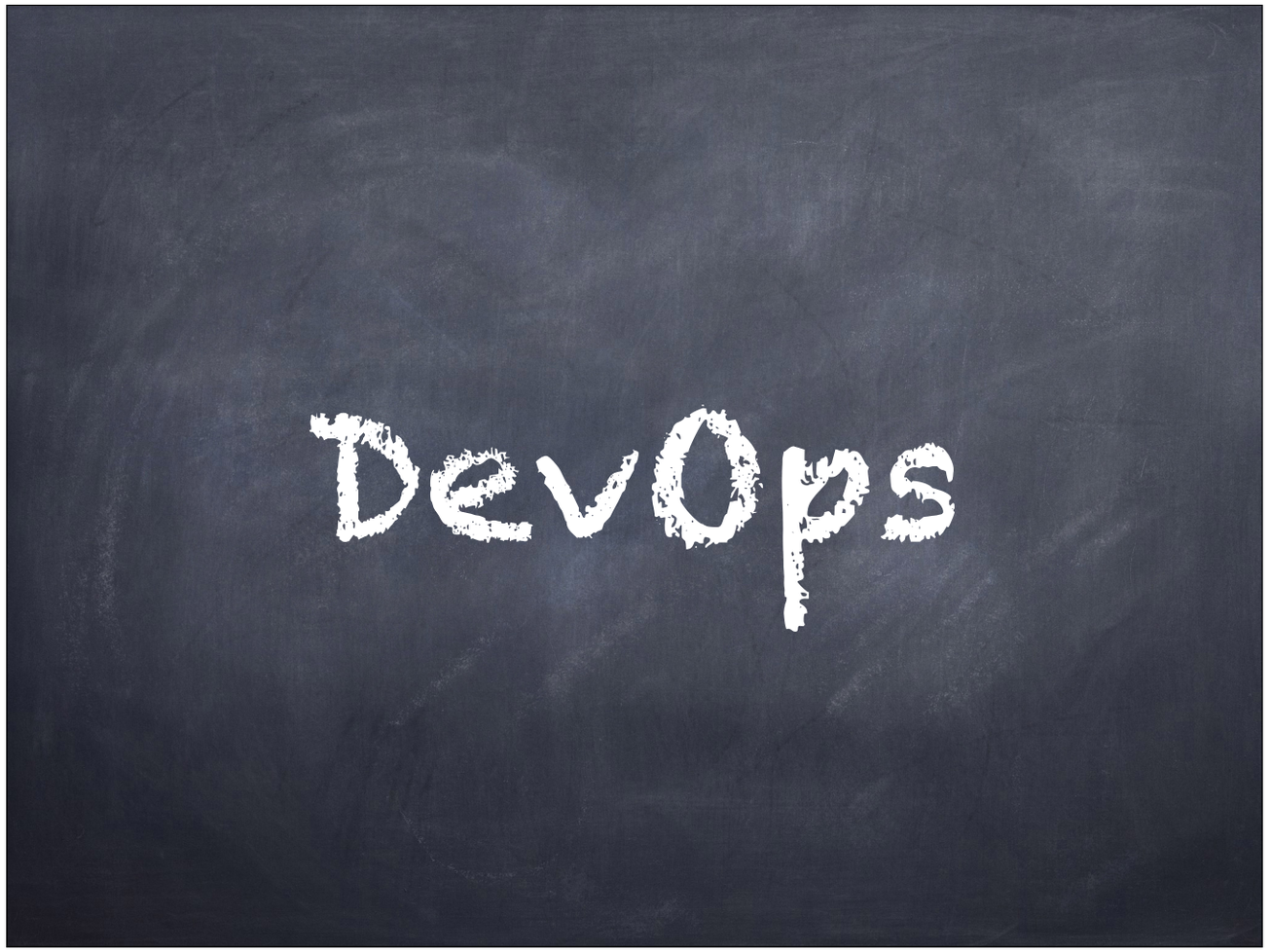
All these amendments of the Agile manifesto improved and change my engagement with customers by introducing the elements of software development proficiency and professionalism.

Instead of shifting all my efforts to satisfy ALL customer needs, I also started to think about the quality of my work and **how the lack of this quality can affect the deliver** of the software which I create.

At the same time, this way of thinking about the software is very hard to introduce in our customers projects. The reason why it is so difficult is connected with **the justification of using all good practices**.

Practices like TDD or BDD, the continuous deployment or monitoring will be always under the constant scrutiny of business people because **they may be seen as a waste of time, because they focus on the software creation, a craft and they don't provide the direct business value.**

I have to admit that I'm not always successful in introducing new ways of thinking about software in my projects. This monster sometimes win the game and recently, it gain also a very strong ally…



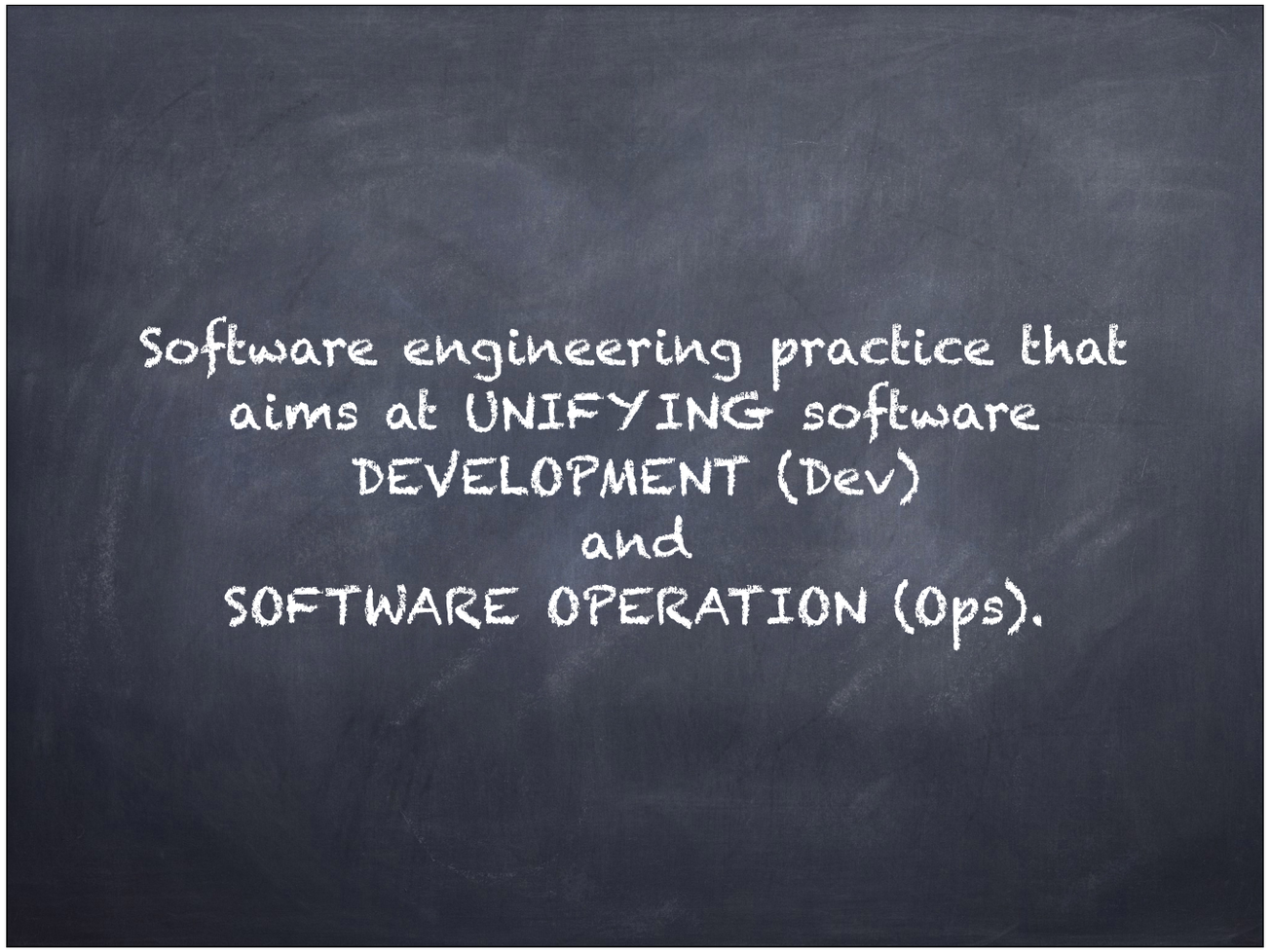
… DevOps.

This monster doesn’t even have manifesto. We only good examples from the industry and a lot of blogs and different books.

In this case, we cannot even say about the vague definition because there is no one, hard definition of DevOps.

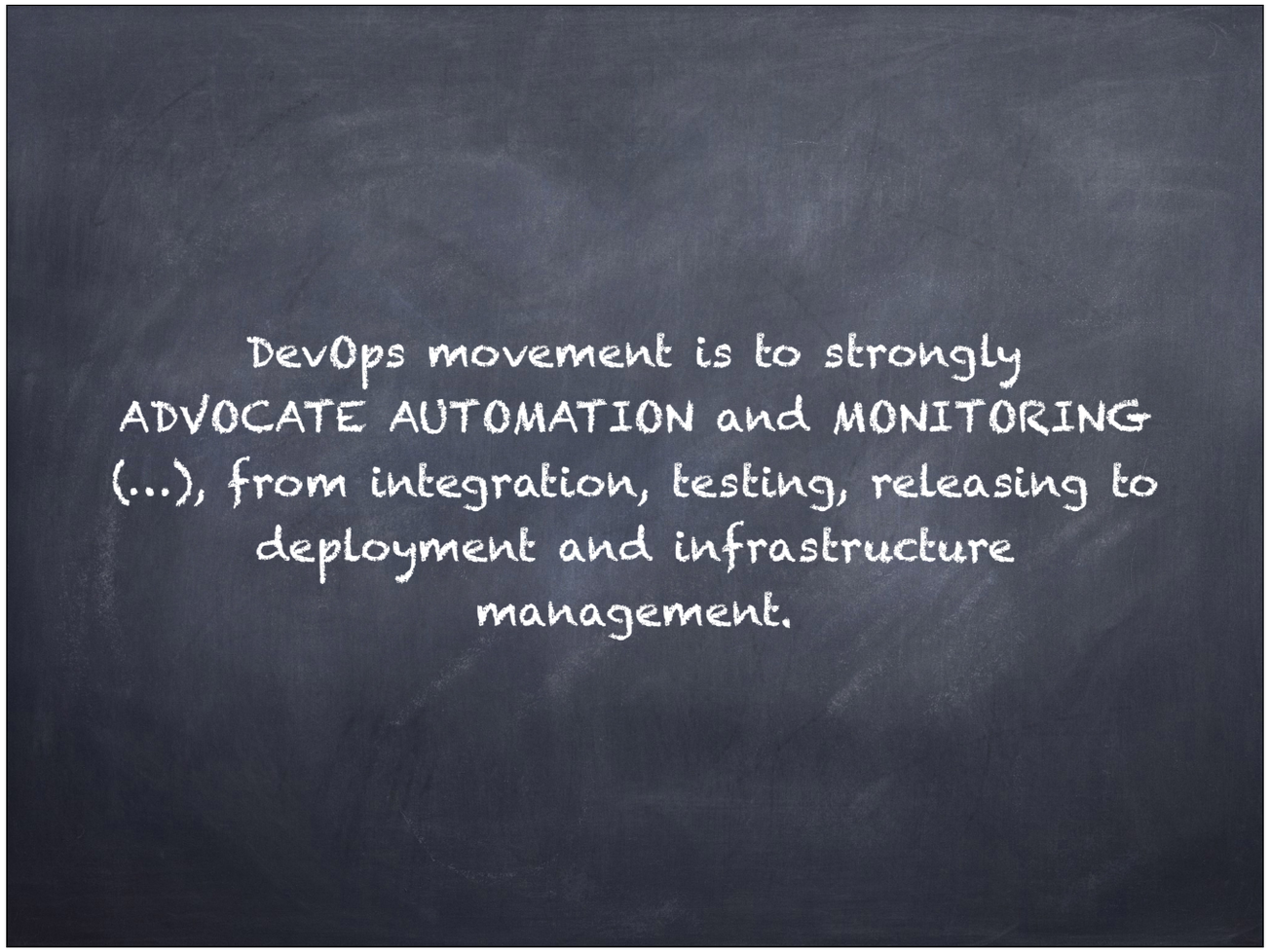
I always like to ask this question. What is the Devops in your opinion? [….Audience…]

I have a separate talk about DevOps and how the perception of this “movement” can cripple your team.



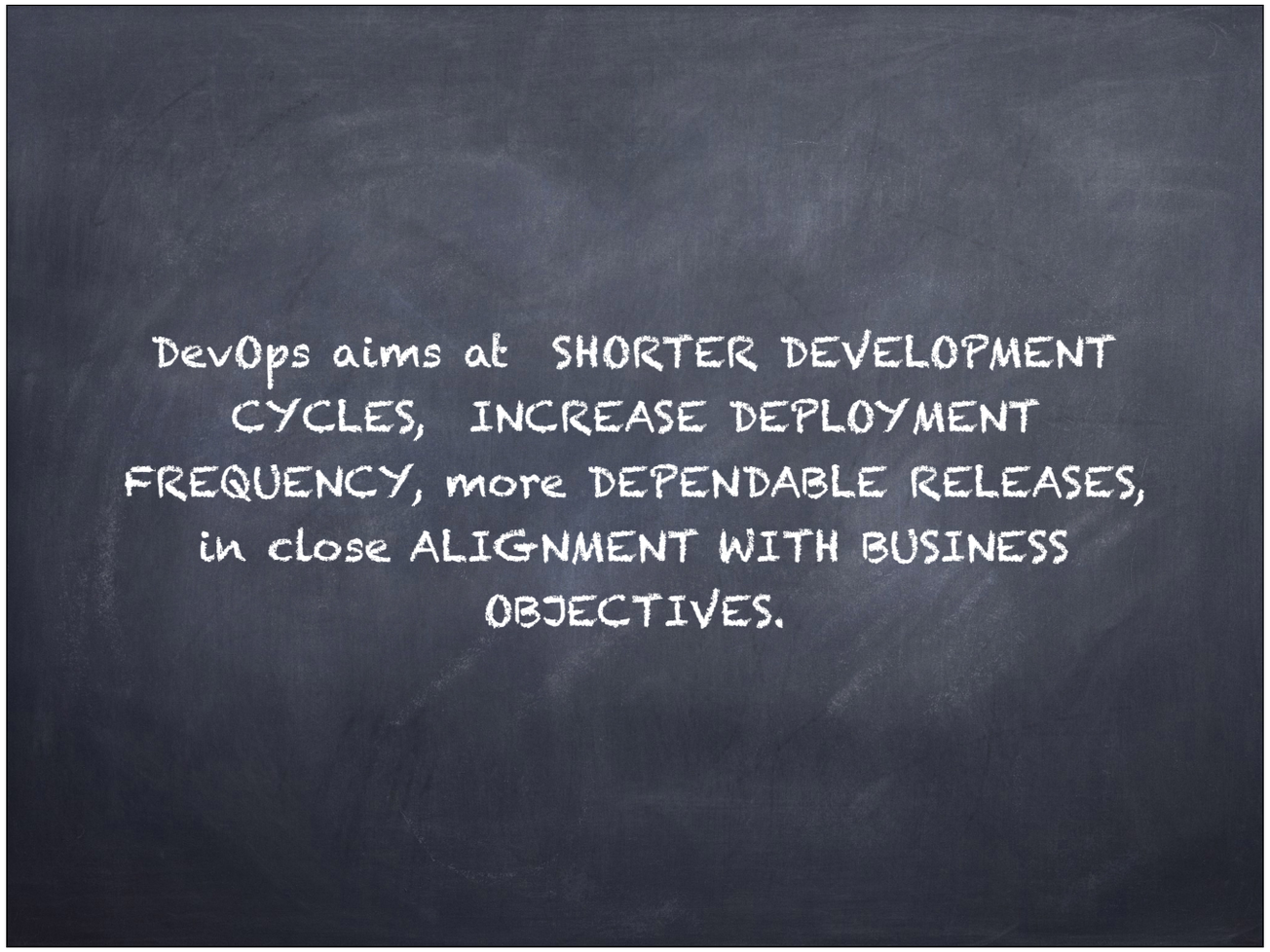
Let’s take a look shortly on this wikipedia’s definition of DevOps.

[READ IT]
What does it mean unifying? Right?



“DevOps movement is to strongly ADVOCATE AUTOMATION and MONITORING (…), from integration, testing, releasing to deployment and infrastructure management.”

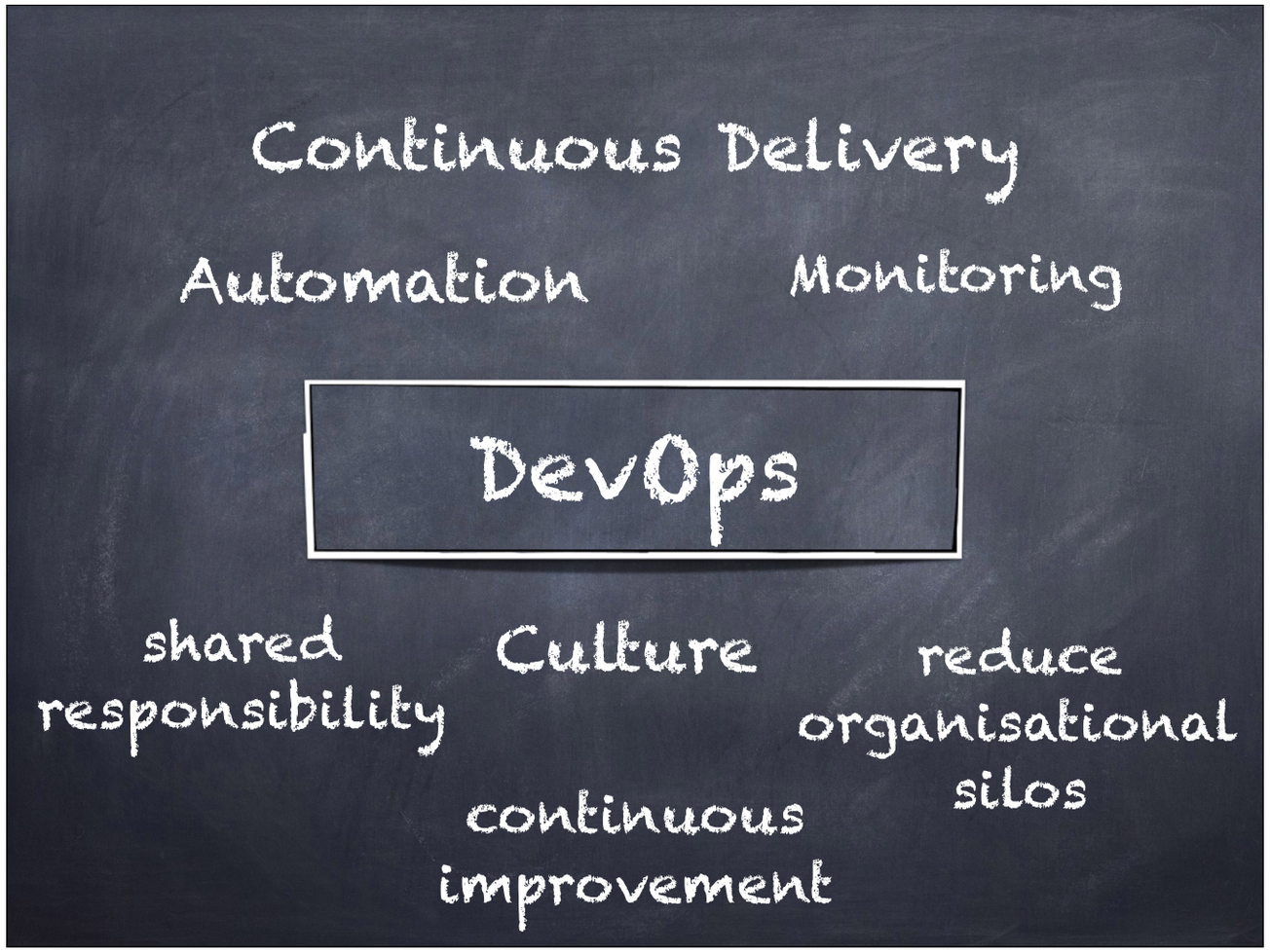
“



DevOps aims at SHORTER DEVELOPMENT CYCLES, INCREASE DEPLOYMENT FREQUENCY, more DEPENDABLE RELEASES, in close ALIGNMENT WITH BUSINESS OBJECTIVES”

Uffff… We get through this dry definition.

After reading that definition, the most common implementation of these words is **“Let’s change the name of the team from Ops to DevOps”. Full stop.**



Ok. DevOps is all about these all things. All these ingredients are necessary to build the proper DevOps.

DevOps is about the continues delivery and short, frequent deployments.
It is about automation of our processes.
It is about the monitoring of your system.

This part is about tools and technicalities.

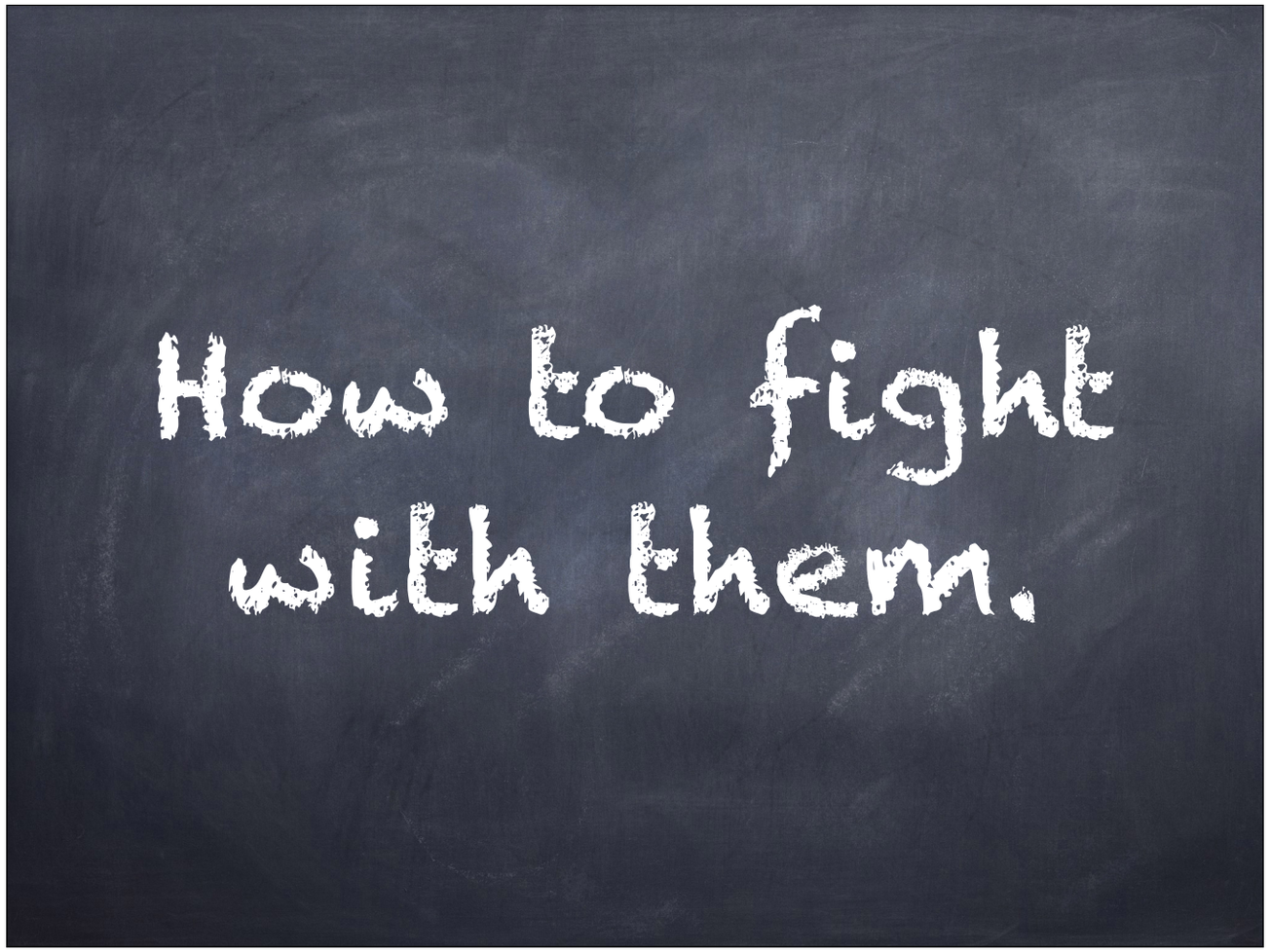
But for me, the most important part is about installing the culture and the environment where

- all team members understand the challenges of software development and IT operations.
- we share the responsibilities for software development and IT operations
- we reduce organisational silos by mutually respect our decisions and experience
- we continuously improve our organisation, our teams and technicalities to achieve our business objectives.

To my disappointment, the second part is neglected in many organisations. DevOps is just a shiny buzzword to create a honey trap for developers and system administrators.

These organisations just renamed teams names from “Ops” to “DevOps” and they still have the same processes, the same mistakes and the lack of continuous improvements.

They introduce new tools without understanding how they influence their project, their teams and their processes.



Ok. We complained a little bit but the questions is “how to fight with these monsters?”.

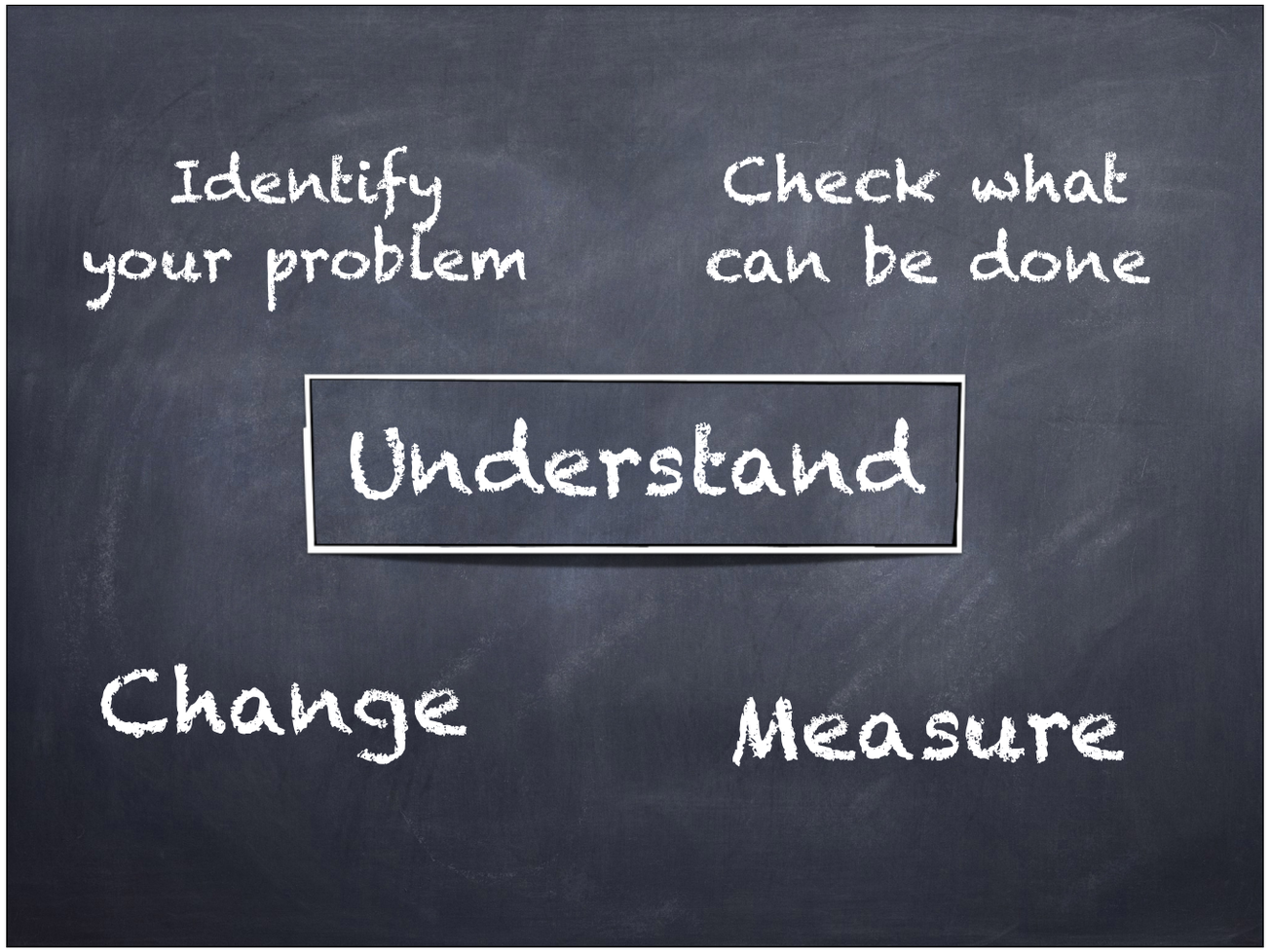
Agile, Software Craftsmanship and DevOps gave recipes for solving the problems in different areas of Software Development.

I propose to create the recipe how to apply them together and in the right way.

I think that it is natural, it is time to create …

… a new manifesto. An IT theory of everything. Our ultimate gun for any manifesto. The past manifestos and the future one.

Let’s call this “Combine Them All Manifesto”. Let’s start this as an experiment.



Understand. The first value.

From my experience, the lack of understanding the core of the problem is strictly connected with all unsuccessful implementations of any approach.

I propose in the first point of our manifesto to focus on:

- **identifying your problem** before your introduce any new practice. Without the proper identification, we are not able to apply any solution.
- check what can be done to fix the problem, find the possible solution and **evaluate each solution in the context of our problem.**
- **introduce a change.**
- **measure results.** This is the most important part of any change is to gather enough data to determine if the change has positive or negative influence.

The is the first point of our manifesto - **I understand what I'm doing.**

The point number two. Embrace the true:

There is no one way to do things.
There is no silver bullets which can fix all problems.

You have to always find your own path because your context, your business is always unique and blindly repeating someone else steps may not be right way of doing things.

Of course, you can find the right steps by **Understanding** your problem**.**

The point number three. Be Courageous to change and improve.

The learning zone is not in your comfort zone. To change things to better you have to leave your habits behind you and explore unknown territories.

You will be lost many times. You will fail and make mistakes. Some of them will cost you a lot but this how we all learn new things.

If you don’t know something, ask for help. Join the right community of a practice. Participate in a training. Read books. Ask experts. And practice, practice, practice.

The point number four. **Continuously improve your process, place of work based on meaningful feedback.**

Improve your process because what works for you today may not work anymore in 2 weeks, 2 months, 2 years. **Measure** constantly different aspects of your process. Find current issues and fix them as soon as possible.

Improve your place of work because you need to create an environment where you can understand your problems, learn new things and continuously improve.

Improve your knowledge because you have to understand better your problems, learn and continuously improve. Software industry is one of the youngest industries and the pace of changes is enormous. You cannot stay in one place.

Gather meaningful feedback of all your improvements because you can measure the effect of your improvements if you have verifiable results. **Do not introduce any change only for a change.**

The point number five.

[This is the hardest point.]

Keep things simple but not simpler because the simple elements are easier to understand, change and deliver. Remember that oversimplification can be equally bad as the over-complication. Both things may reduce the clarity of your solution.

Use small increments to introduce change because these small baby steps will help to understand the nature of the change and the relationships between the change and the behaviour after the change.

The point number six.

Be respectful for other people because they share the same goal with you. Respect their experience and combine it with yours.

Keep good relationship with people in your project because our human nature make the communication easier when people are happy to work with each others. Make peace not war.

This point may not be obvious in the first place. I, personally, was lucky enough to work always with open-minded people who respect my lack of knowledge and they allowed me to fail but

I've also met a few people who have a very destructive influence on my teams. The toxic relationships may destroy your motivation to make things better, destroy your teams or even companies. Creating an environment when our creativity and motivation thrives is extremely hard under pressure which we are facing in our projects but we have to remember **our approach**, **our behaviours** shape the foundation of good working places.

Here it is. Let’s create a web site and start collecting signs.

Maybe, this manifesto is just an experiment. Maybe it is just a silly attempt to control the uncontrollable software development process.

Even if it is a little bit silly, I would like always apply these rules in my projects. I would like to use these rules whenever my team, my organisation or my project introduce Agile, Software Craftsmanship or DevOps techniques.

I would like to ….

… build future by learning from the past.

Because all points from our new manifesto were derived from the roots of Agile and Software Craftsmanship movements.

Extreme Programming. A forgotten set of good practices which shaped Scrum or Kanban.
You can find all XP values in our new manifesto. The were created at the end of 20th century/ the beginning of 21st century but all these values are still valid, in opinion.

We don’t have to find new ways or new solutions. We should move back to point zero to understand why are we develop our software in Agile and DevOps style.
Plenty of things were discovered and described in 60’s and 70’s.

We don’t even fully change our worst enemy…

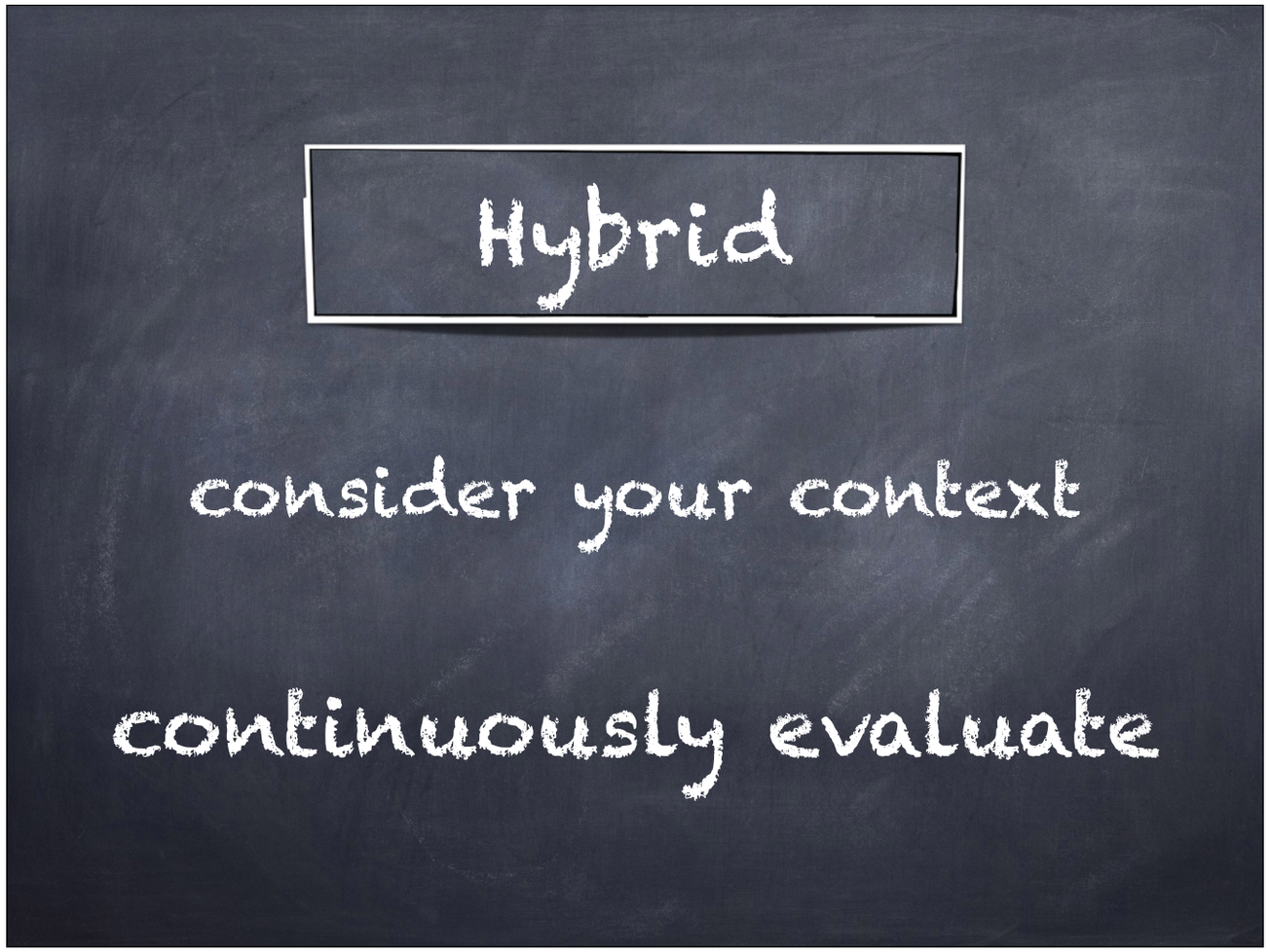
Waterfall. Some parts of Waterfall you can find in our Agile and DevOps environments.

We know that this way of creating a software is not sustainable but we didn’t throw away everything.

Scrum is small iterative version of waterfall (of course with some adjustments). We create backlog to gather our requirements. We decide about the most important elements and we design our product. During the sprints, we are implementing the feature and we verify them by testing our software on many different levels. And finally, we maintain our systems by reacting on changes and fixing bugs.

All these elements existed before Agile. Agile, in my opinion, was not a revolution. It was an evolution.

What is the next step in this evolution?



In my opinion it will by a hybrid, a mix of different methodologies, practices and approaches.

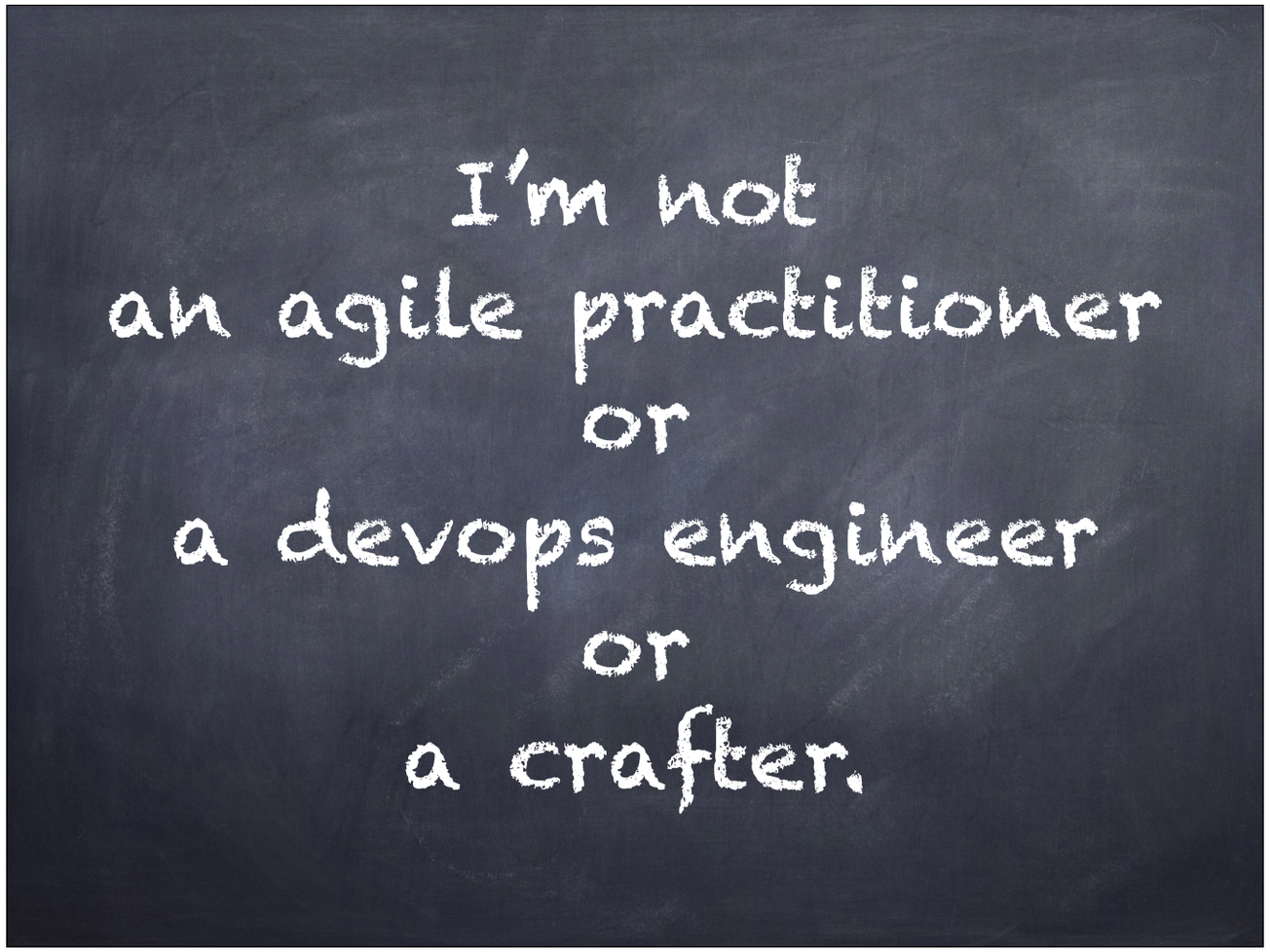
Because:

- Business is not always open to cooperate.
- Developers are not always open for change.
- We face the pressure of time in highly competitive industries.
- not every company have good culture of continuous improvements

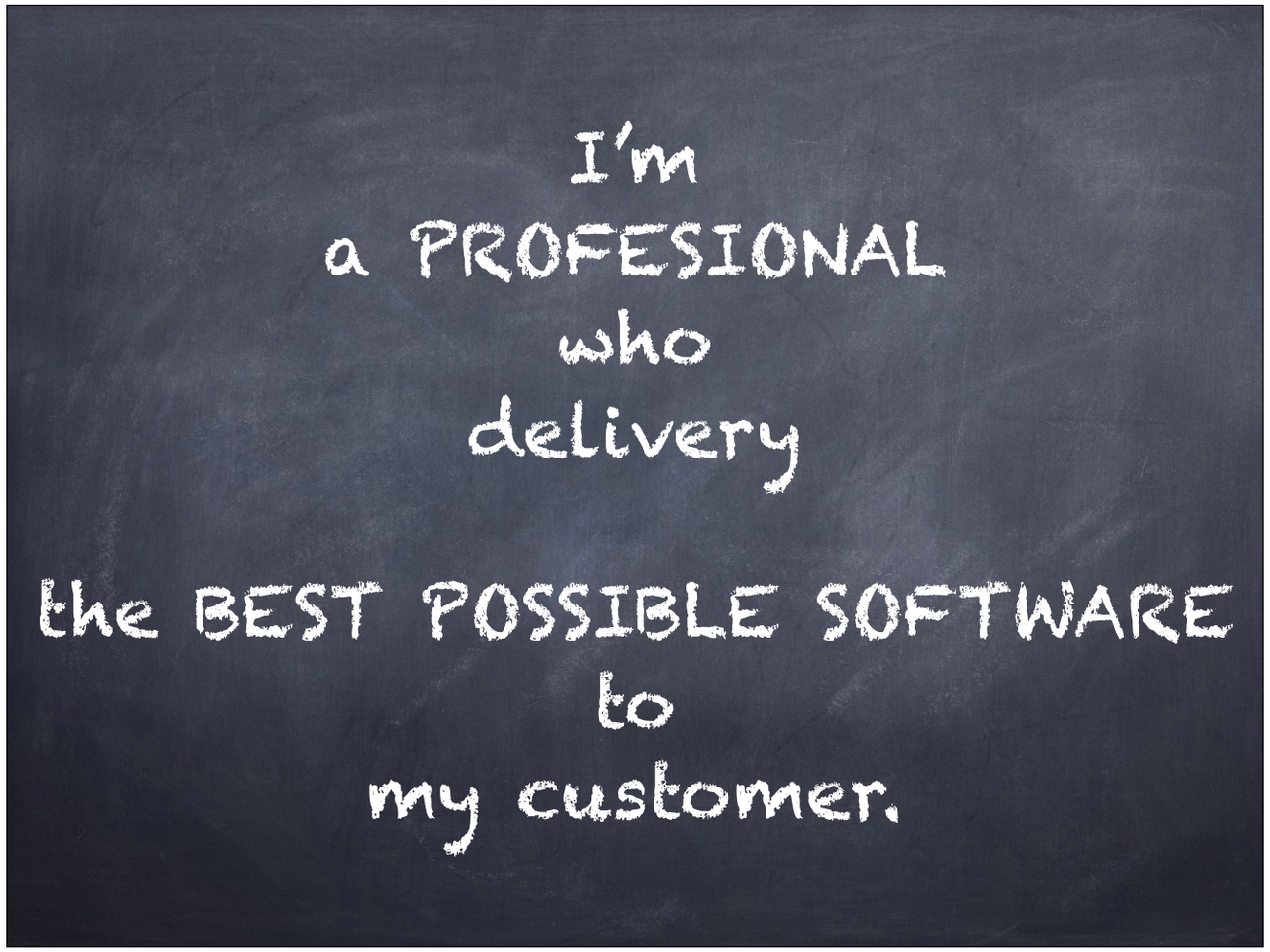
we have adopt our approach and choose elements which we can be introduce in our unique context.

In the hybrid model we will be forced constantly evaluate our decisions and adjust them to constantly changing world.

In the hybrid model…



I'm not an agile practitioner or a devops engineer or a crafter.



I'm a PROFESSIONAL who delivery
the BEST POSSIBLE SOFTWARE to my customer.


Thank you