It's All Upside Down

What I've learned about software development and why it seems opposite to everything I was taught

Paul E. McMahon

It’s All Upside Down

What I’ve learned about software development and why it seems opposite to everything I was taught

Paul E. McMahon

This book is for sale at http://leanpub.com/itsallupsidedown

This version was published on 2017-05-07

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Introduction

I have been involved in the software business for over 40 years and during this time my views on how to help software teams succeed has evolved so much that what I recommend to many clients today often seems completely opposite to the fundamentals many of us were taught. By "fundamentals" I mean principles such as

- find defects early
- understand your requirements before you design
- test before you release
- plan and define your process before you implement
- and so on…

However, I don't believe what I recommend is actually opposite to these principles at all. It just seems that way because what many of us were thinking when the fundamentals were first explained to us has turned out to be quite different from what actually works.

I would also like to point out that I am not the first person to have observed this phenomenon, and it is not unique to software development. As the immortal Yogi Berra once said, *"In theory there is no difference between theory and practice. In practice there is."*[1]

I don't believe this situation implies there is necessarily anything inherently wrong with many long held software

[1] http://c2.com/cgi/wiki?DifferenceBetweenTheoryAndPractice

engineering fundamentals. But there is a very real problem centering on common misbeliefs related to **how to** best achieve many of our well established software engineering guiding principles. These misbeliefs are tied to the **activities people think they need to conduct**, "**how much**" of those activities they need to conduct, and "**when**" they need to conduct them, to effectively implement software engineering fundamentals.

What is in this book

In this book I provide true software development stories that may challenge long held thinking. I highlight **26 upside down principles** along with upside down principle **clarifying thoughts**. At the end of each story you will find **extended clarifying thoughts** in the upside down principle summary sections. I also highlight 18 coaching tips that can help you get your organization "right side up" with respect to performance.

If you are wondering if a book that highlights coaching tips is for you, it is. I believe everyone in an organization should view themselves as a coach. In order to help coaches everywhere, in Part II of the book I have framed the highlighted principles and coaching tips that have materialized from my stories within a framework I like to use called Essence. I find Essence to be a remarkably straightforward medium for communicating software development practices and fostering collaboration, even among non-technical stakeholders.

If you have not heard of Essence yet, you have probably heard of the foundation from which it evolved, which I explain further in Part II. I will explain more below how Essence relates to my stories and coaching tips. If you choose to read

Part II, I think you will agree that Essence is quite useful for capturing and describing these lessons, tips and best practices, and I hope you will be able to find ways to make Essence useful in your own work.

About Essence

In the stories in this book you will learn many questions that I asked my clients and coaching tips I gave them to help them overcome common obstacles and achieve higher performance. Many of the questions I ask and the tips I provide are based on experience. But what if there was a way to share these questions and tips with practitioners in a form they could easily access when a coach isn't immediately available?

I have been involved since 2010 in the development of a new framework, referred to as Essence [1, 2, 3, 4], that provides a way to do just that. I sometimes refer to Essence as a "thinking framework," but, in fact, Essence is more than just a "thinking framework". Essence provides a "common ground" set of essentials relevant to all software development efforts. This includes a set of checklists that can stimulate the kinds of questions that I ask my clients. You will learn more about these questions in my stories in Part I of this book.

Essence also includes a way for anyone, including consultants, coaches, practice/method authors and practitioners, to express their practices and tips in a common language highlighting the unique discriminators of their approach. This way of expressing practices in the Essence language is referred to as "essentialization."

In Part II of this book I provide an introduction to Essence, and a simple example of an "essentialized" practice that is similar to a practice I created for one of my clients discussed in Part I of this book. I also provide in Part II a summary of the 26 upside down principles in an essentialized form. You will also learn in Part II how anyone can use the Essence language to capture the questions that should be asked– in a checklist form– and the activities recommended to be conducted, such as those I share in Part I of this book.

How to use this book

I highly recommend that you read all the stories in this book straight through from beginning to end. As one of my reviewers commented, the book is short enough to be read in 3 hours. Then, use Part II to review the principles and learn about the Essence framework including the checklists related to each story. A cross-reference is provided in Part II from each essentialized principle back to the relevant story in Part I.

Another way to use this book, for those who don't have time to read all the stories, is to jump directly to Part II where you can learn about Essence, review the principles and related checklists, and then jump back and read just the stories you are most interested in.

Why I wrote this book

This is not a book about theory. The stories I share are all true. I wrote this book because I believe the stories, together

with the principles, principle clarifying thoughts, coaching tips and related Essence framework checklists can stimulate deeper reflection and discussion helping you discover your own best "how to" software engineering approaches within your organization.

About the stories in this book

The stories in this book demonstrate what I have learned about how predictable performance is achieved in organizations today. In the past we have spent significant effort training our people by focusing on "defined processes"[2] and "best practices"[2]. But true stories of what has been proven to work are better at communicating "**how much**" of certain activities need to be conducted, "**when**" those activities should be conducted, and "**exactly what activities should be conducted**," which is what most practitioners want to know.

Sharing stories works because when people hear true stories about how another team handled a problem similar to their own they can often quickly translate that experience to their own situation. "Defined processes" and "best practices", on the other hand–specifically the way they have often been communicated in the past– leave too many important questions unanswered. Once you understand this fact, it can lead you to think differently about many of our long held fundamental principles. Let me give you an example.

[2]Some organizations use the term "process" and others use the term "practice" to mean the written description of how they would like their people to behave. But the practices (or processes) that people actually follow may not be written down (i.e. they are tacit). In this book I use the term "practice" and "process" interchangeably and I mean it to include both types—written and tacit.

Do we really need to define our processes?

A fundamental many of us were taught is that we need to define our processes and train our people in our processes prior to using the processes on a real project. We have also been taught the importance of ensuring our teams are using our defined processes in a consistent repeatable way before trying to improve them [5]. In other words, many of us have believed for years that we need to focus on defining, training and achieving repeatable basic processes within our teams before we focus on raising the performance of our team.

Now, I understand why this fundamental notion seems to make sense in theory. But let me tell you a little bit about what I now do with most of my clients that seems upside down from this fundamental principle– but has proven to work.

First, I spend very little time defining a process before I ask my client to try it out by using it, and I often encourage them to use it on a real project, rather than in a pilot environment. This is clearly not what I was taught to do. Now don't misunderstand me. I am not saying you shouldn't define your processes and train your people before asking them to use the processes. Nor am I suggesting you should try out completely unproven ideas on critical projects before you have any idea at all if they might work. You will learn more in Story Two of this book why I have found this approach works best in practice.

But now let me share a related idea to help you understand some of my thinking. What I have learned is that most companies today that are doing software development and

know they have weaknesses and want to improve already have a way of working that has strengths as well as weaknesses. It may not be a repeatable institutionalized process that is used consistently across the entire organization. But there almost always exists, even in organizations that operate almost entirely in an ad hoc manner, at least one project, or group of people on a project, that have established their own way of working that is working well and proving successful. This has been my experience.

When organizations begin asking questions about improving their way of working, or even considering trying something completely new, it is usually because they know they can do better. And many of them know they can do better because part of their organization actually is doing better and they are ready to start using those proven best practices right now so they can perform better too.

Once organizations arrive at this point, what I have found to be most effective to help them move forward is two-fold. First, they need some simple guidance and coaching in how to capture what is already working well– even if it is only for a small group in their organization– and spread it to others in their organization. Second, they need help identifying where their most critical weaknesses lie that are hurting their performance, and they need help figuring out the most important changes they can make right now to their current way of working to address these clear weaknesses. It is often in this area where critical weaknesses exist that the best opportunities to try something new are found.

Upside Down Principle One:

Traditional thinking: Plan and define your process before you use your process.

Clarifying thought: Conduct just enough planning for your team to get started moving forward.

More reasons not to spend a lot of effort defining processes before using them

I have also found that it can become a wasteful exercise to spend more than a minimum amount of effort documenting processes that have not already been proven to work successfully in your specific organization. For this reason, I have moved away from putting much effort into process definition up front. The best time to put significant effort into documenting processes is after–not before– we know what works. On real projects every day teams learn and every day those teams should feel empowered and encouraged to improve the way they work based on what they are learning on the job. Unfortunately, what happens too often is that organizations spend far **too much effort** trying to define their processes in detail before their teams actually use and prove out those processes on real endeavors.

This strategy inevitably leads to internal organizational struggles between those who are being asked to use the processes

(e.g. the practitioners on the job who are trying to develop quality software for their customers) and those who defined the processes based on what has come to be referred to as "industry best practices." Let me give you another example.

Are industry "best practices" best for you?

Now, once again, don't misunderstand me. I have nothing against industry best practices. In fact, a large part of what I do with my clients is sharing industry best practices trying to enlighten them when I see weaknesses in their current way of working where what other organizations have learned could help. But the problem is, while many common misapplications of fundamentals repeat in the software industry, the answer can never be found completely in any single "best practice" solution proven somewhere else in a different organization than your own. This is not to say you can't learn from the experiences of others. But it is to say your situation will always have your own **unique conditions and constraints** that your organization needs to think about as you consider lessons and best practices from other organizations and industry in general.

Stated another way, while industry best practices and lessons are great, they do not alleviate each organization from listening to their practitioners, understanding their needs and understanding the specific needs and constraints of their stakeholders, environment and their product.

Upside Down Principle Two:

Traditional thinking: Focus your improvement efforts on industry "best practices".

Clarifying thought: Focus your improvement effort on your team, stakeholders and environment.

Is process or performance more important to you?

We have for a long time been hearing about the importance of defining your process first, sharing industry best practices, and sharing lessons learned as keys to improvement. While I do not dispute the value in these proven approaches to improve performance, the way they have been implemented in the past has led many organizational improvement efforts to fall far short of their goals. It is my belief, based on my 40 plus years of working in this industry, at least part of the reason that the software engineering community continues to face this problem is because the emphasis on how to go about improving performance has been significantly misplaced.

Performance itself should be the primary focus. Process is secondary. Unfortunately, the tail has been wagging the dog far too long and this is at least part of the reason why I titled this book: *It's All Upside Down.*

Summary Upside Down Principles 1, 2

Principle One: *Plan and define your process before you use your process*

Extended clarifying thought: Conduct just enough planning and definition for your team to move forward using the key processes needed to get started so they can prove them out, and refine them with improvements that work best for their situation.

Note: This is an example of what I meant when I referred to the need to communicate "when" certain activities and "how much" of those activities need to be conducted.

Principle Two: *Focus your improvement efforts on industry "best practices"*

Extended clarifying thought: While industry best practices and lessons are useful, they cannot replace listening to your practitioners, understanding their needs and understanding the specific needs and constraints of your stakeholders, environment and your product.

Story One: Do You Really Need Measures to Improve?

A fundamental principle many of us were taught is that we need measures to improve. We've all heard statements such as:

"You can't manage, if you don't measure" and

"If you can't measure what you are doing, you don't know what you are doing" and

"Without measures you will never know for sure if you really improved"

These statements on face value seem hard to argue with. So many organizations that are trying to improve performance spend significant effort putting a measurement program in place before they think seriously about improving the way they work.

A measurement program can be a good and useful thing if it's set up in a way that works for the organization. But before we talk about that, let's step back and ask the question: Do we necessarily need a measurement program to address a problem that is obvious to the organization?

Should you really set up a measurement program right now?

Now, please understand my point. To set up a measurement

program the right way takes considerable effort up front. First, you need to look at your own business and, specifically, your business objectives. You need to ask:

"What are our goals?"

Then you need to ask:

"How will we know if we are on track to achieving our goals?"

Then you need to ask:

"What can we measure to help us in this assessment?"

When organizations start down this path often it requires multiple brainstorming sessions and discussions about measures which frequently end up flowing down into the organization where derived measures need to be gathered, aggregated, and analyzed periodically. Often this effort takes significant resources away from direct project activities that create revenue for the organization.

So let's now step back. I am not here intending to argue with the value of setting up a measurement program. If you are considering setting up a measurement program in your organization that might be a great thing to do. But I just want you to think a bit about where your organization is, and if that is the best area to expend energy right now given where your organization is. Now I want to give you a simple example to help you as you think about this.

Does your organization suffer from the common problem of constant "interrupts"?

A common problem I often see in client organizations is the software team constantly dealing with interrupts due to the latest crisis in the organization. Constant interruptions can create significant disruption degrading any chance for the

team maintaining a predictable velocity. This particularly affects organizations that are trying to implement agile/Scrum practices, which require the team to plan and commit to a fixed iteration length and an agreed to set of backlog items to complete.

One approach that I have seen used in an organization to try to address this problem is for the team to start measuring interrupts. I will refer to this organization as NANO. The thinking is that if we can gather real data that shows management the impact of the interrupts, then management might understand why it is so important that we do something to change the interrupt driven culture.

On the surface this sounds like a great approach, and I have heard many people agree that the idea is a good one. I have also observed multiple organizations take this approach to help bring to light this important issue they would like to solve. I have even seen one organization go so far as to have software team personnel enter a ticket in their ticket system that allows them to identify each interrupt, how many hours they spent on the interrupt, and a brief description of the interrupt.

This data allowed the interrupts to be analyzed and grouped into categories. Then a graph showing the cost of various categories of interrupts could be created and shared with management or any other personnel who might be able to take action to potentially help change the interrupt-driven culture of the organization.

Now, understand that I think this is a great idea. Measuring team interrupts could even be a derived measure in a measurement system where one of the organizational agreed to business objectives is to achieve predictable product releases.

Such a measure could potentially help contribute to achieving this goal by bringing to light a problem that is hurting the organization's performance, related to achieving predictable releases. An organization could even decide to use the interrupt measures gathered to predict future performance by building into their plan anticipated interrupts based on past interrupt frequency. But this leads to an important question:

Is measuring interrupts the best approach to solve the common interrupt problem we see in many organizations?

A different approach to solve the common problem of "interrupts"

Let me now share another approach that I observed a different organization take that allowed them to solve their interrupt problem in a surprisingly short period of time without first measuring the frequency and impact of their interrupts. This organization I will refer to as NORO. NORO had a culture of always reacting to the latest fire-drill by throwing everyone on the problem as soon as it occurred. Interrupts could come to the software team from various sources and at any time. The accepted culture in the organization was for the team to drop everything and immediately solve the problem especially when the interrupt came from the VP of Operations, or from one of their key customer support people.

I was hired by NORO because they knew they needed to put some process discipline in place, and the recognition of this need started at the top of the company with the president who hired me. When I began to understand how this organization was operating, it quickly became clear to me that we needed to start with basic management practices. I conducted basic Scrum training for the software team. During this training I facilitated a risk brainstorming session to get out on the table

any concerns people had that could derail the use of the new Scrum practices.

During this brainstorming session, the Scrum Master, whom I was training, said that the biggest risk he saw was that we were only training the software team. He expressed concern that the rest of the organization would continue to work the same way they had always worked. He was concerned that the software team would never be able to succeed in their planned sprints, if the interrupts continued.

Strangely, once we began the sprints the interrupts seemed to vanish. At the first sprint retrospective we discussed why we seemed to have solved the big risk of interrupts. The Scrum Master reasoned he wasn't getting interrupts because most of his interrupts came from a key customer service representative who had been tapped to become the product owner. Because that customer service representative understood his new role as product owner, and his responsibilities with regard to all work going on the product backlog and being prioritized before worked on, he understood the new way of working and had stopped with his interrupt behavior.

The president of the company had also taken it upon himself to learn the new process and had attended the Scrum training. He also participated in the first sprint retrospective. During that retrospective he said that the reason the VP of Operations wasn't interrupting anymore was because he had been keeping the VP up to speed on the new way of working and what the team was working on.

What everyone realized was that we solved the interrupt problem by solving the real reason people interrupted. We put a new way of working in place, and we refined it on a real project. Because the effort was supported by the president and

key influential stakeholders, the employees understood that the company was serious about operating differently and so the new behavior just happened. We didn't need to measure the cost of interrupts first to prove what everyone knew was hurting performance. Everyone knew the interrupts were hurting performance so they just all agreed to change their behavior.

Upside Down Principle Three:

Traditional thinking: You need to establish measures first to be sure any process changes lead to real performance improvement.

Clarifying thought: Sometimes it's easier to just fix a problem than worrying about how to measure the effectiveness of your solution.

What happened in the organization that measured their interrupts?

In the NANO organization where they measured the cost of interrupts some interesting things happened. When we analyzed the interrupt data we found that many of the entries people had logged as interrupts didn't sound like activities that should have been treated as interrupts. Many of them were normal activities that could have been predicted and planned for, if the team had done a more thorough job of thinking ahead of time about all of the work that would need to be done in the upcoming sprint. I am going to give you an example below.

The first lesson learned was that the team had been taking too narrow a view of the scope of work to be done in the upcoming sprint by just looking at the items on the product backlog, rather than including all work that was likely to be needed to be done. This effort also uncovered the fact that

many roles in the organization needed to be reviewed from the standpoint of responsibilities. This occurred because what some people were calling interrupts should have been viewed as a normal part of their job responsibilities.

For example, a technical lead who was also a developer was viewing requests to review a developer's work as an interrupt. So the interesting thing we learned by measuring interrupts at NANO was that there weren't really as many interrupts going on as many people thought. People needed refresher training in their own responsibilities and how to estimate their capacity and effort required to complete certain tasks, given those responsibilities. You will hear more about how we helped NANO with their interrupt problem later in this book.

Clearly, if you can figure out the root of the problem and just solve it as we saw in the second case, it is the ideal approach. But in many organizations this isn't as easy as it was at NORO. In many organizations where I have seen interrupt data measured and shared with management, too often no action results to change the interrupt behavior.

What else should you do, if you decide to measure interrupts?

In some cases measuring may be the best you can do, but if you take this approach I suggest you capture some information about what the interrupt is and how much time the interrupt is costing you. Then you should analyze the interrupts and see if you can categorize them into certain types ("or buckets") of interrupts. This might lead you to develop a graph where you can see which categories are costing the organization the most.

Don't be surprised if you find yourself refining your cat-

egories as you conduct this analysis. But my suggestion is don't stop with just analyzing the data. You should go further. I suggest you talk to people in your organization, and hold some brainstorming sessions to come up with possible approaches to solve the interrupts—especially the most costly ones– before you take the results of your analysis to management.

You might even suggest that people try some of the ideas you come up with on real projects. Showing management data that can be used to explain what is going on in the organization is good, but bringing a proposed solution is better. And it is even better yet, if you can point to a real project that is using your proposed solution as evidence that it works.

Upside Down Principle Four:

Traditional thinking: Showing management hard data related to a problem is the best way to get them to support a needed process change.

Clarifying thought: Showing management a proven solution is better than showing them hard data about a problem.

Why the interrupts vanished at NORO

In NOROs case the interrupts went away without measuring them because they implemented a way of working that eliminated the root cause of the interrupts. Note here that I use the phrase "way of working" rather than "process". This is intentional because for many people the word "process" implies a "documented process" that has been reviewed, approved, placed under control, and trained in a formal manner.

Rather, when I say "way of working" I am talking about a set of activities that people use because they recognize how those activities can achieve a desired result. In the NANO organization we learned that the interrupt problem could be substantially reduced through on-the-job coaching of responsibilities associated with certain team roles and coaching related to effective sprint planning. In the NORO case when someone came with a potential interrupt, it wasn't *"drop everything and solve it immediately"* anymore. It was

"Go see Jim, and he will place it on the backlog."

Soon everyone at NORO understood that this was the new way of working. Because people understood the new behavior with the new way of working, the interrupts just stopped happening.

Summary Upside Down Principles 3, 4

Principle Three: *You need to establish measures first to be sure any process changes lead to real performance improvement*

Extended clarifying thought: In general measuring is important to be sure you aren't fooling yourself, but sometimes it's easier to just fix the problem when everyone agrees and you know how to fix it.

Principle Four: *Showing management hard data related to a problem is the best way to get them to support a needed process change*

Extended clarifying thought: Showing management hard data related to a problem, along with hard data that demonstrates a real solution that has been proven to work is the best way to get them to support a needed process change. Often, in cases where the problem relates to interrupts, solutions to root causes are found in helping team members understand the full extent of their responsibilities.