## Context - Introduction

### Why did you pick up this book?

When trouble comes, think of yourself as a wrestler whom God, like a trainer, has paired with a tough young buck. For what purpose? To turn you into Olympic-class material. But this is going to take some sweat to accomplish. – Marcus Aurelius

----

*To pursue the impossible is madness – Marcus Aurelius*

*----*

*Remember that our efforts are subject to circumstances; you weren’t aiming to do the impossible.*

*Aiming to do what then? To try. And you succeeded. What you set out to do is accomplished.*

*- Marcus Aurelius.*

This book is called “Delivering the impossible.” This is quite a provocative title. So, I am interested, why did you start to read it?

Have you been given the task of delivering something that you think is impossible? If so, did you pick this book up with the hope that it might help you do that? Have you given someone else the job of delivering the impossible? Did you pick up this book so that you could give them some hints and tips?

But that isn’t the real question I’m asking. The real question I’m asking is why did you pick up a book that has a contradiction, a nonsense even, as a title? Because… does it need saying? Delivering the impossible, is, well, impossible.

Here, in the first few pages, you might already be losing patience. You might be thinking “yeah, yeah, yeah, this quibbling over words is all very well, but we both know why I’m here.

“My project is in big trouble, I have literally no idea what to do to save it, or even what to do to make it slightly better. I’m working long hours and I’m having sleepless nights. Can you help me with that or not? If you can, great. Let’s get to it. If you can’t, I’m out of here.”

OK, OK, I’m sorry.

If you’re that desperate, if you’re project is really in that much trouble (and believe me, I’ve been there) then forget about all this and go straight to the Concrete Practice section on page [NNN].

Answer the questions that you find there. They will help you get a better picture of just how “impossible” the situation is that you’re in and what (if anything) can be done to make it better.

Follow the suggestions for what to do, based on your answers to those questions. Don’t worry. Help is at hand.

If you didn’t scurry off to the Concrete Practice section, and you’re still here, I think I might have an answer to the question of why you picked up this book. Even, perhaps especially because of its paradoxical title.

I think that “Delivering the Impossible” sounds like an exciting and enticing prospect. I also think that we’ve all been in the position of being asked to deliver something that we simultaneously know that we can’t deliver and would also dearly love to be able to deliver. We’ve held those two thoughts roughly simultaneously in our heads: “This is impossible!” and “I’d really like to get this done.”

I also think “Delivering the Impossible” sounds a lot more enticing than “Delivering the Possible” which, strangely, sounds a bit drab and mundane. And this is of course a paradox, because only possible things can be delivered. Only possible things can ultimately deliver value and success.

But you, the reader, if you’re the kind person who is used to getting involved in paradoxical conversations, might realise that it’s always a good trick to fire any difficult question right back at the questioner. When asked “why are you interested in a book called ‘Delivering the Impossible’, you might just turn the question straight around and ask me, why am I writing a book with such a puzzling title?

I’m Mark Stringer. I’ve been working in software development since 1994 and I’ve been managing software development using what are called Agile methods since 2009. Not long after I started using Agile methods to deliver projects, I started to realise that there was something contradictory and nonsensical about the way projects were talked about.

### ### Back to the Plot

This book is called “Delivering the Impossible” to be totally up front about the contradictory nature of managing and delivering successful projects.

It’s the general outlook of this book that project management is riddled with contradictory, nonsensical and conflicting ideas, many of which cannot be resolved, but only managed.

In recognition of this, I’ve put one of the contradictions right on the front cover.

The main idea of this book is that understanding one particular contradiction makes delivering projects that seem impossible, possible.

Here’s the contradiction that I’ll talk about most:

1. Projects get funded using a particular process (I’ll call it a value stream, the “virtual value stream”). Another way of thinking of this value stream is as the P2C stream – powerpoint to cash. The dream stream.
2. Projects make money, or give other kinds of value to their users, using a different process (I’m going to call this the “virtuous value stream”), which in software development and crucially unlike in many other, preexisting, industries has to be built, discovered, engineered. Another way of thinking of this value stream is as the C2C stream – the code to cash stream. The cream stream.
3. Making a good attempt at doing 2 – (the virtuous value stream, the code to cash, the product) attacks, criticizes, undermines and derides 1 – (the virtual value stream, powerpoint to cash, the dream).

Why?

Where is there a contradiction? Why does 2. Attack and undermine 1?

There are two main reasons.

The first reason is that attempting to do things in reality throws up lots of problems that just don’t get thought of in the idea. In the idea, everything is perfect but in reality, Murphy’s law – “anything that can go wrong, will go wrong” applies to the real world.

When people talk about Lean, they talk about mapping value streams. But if you’re building a thing that’s entirely new the value stream isn’t there to be mapped. The value stream needs to irrigated. The value stream needs to be built.

A picture containing text, whiteboard

Description automatically generated

As (probably) Alan Kay said:

“point of view is worth 80 IQ points.”

The aim of this book is to explore a point of view which sees software development in terms of these two value streams:

The virtual value stream – powerpoint to cash. This is where value and money is given to an idea.

The virtuous value stream – working code to cash (or other kind of value for its users and owners). This is where value and money derive from the actual use of the working software.

In between those two streams is a desert – which is where most projects spend most of their time.

Thinking about software development in these terms offers one explanation why managing software development is so conflicting, contradictory and hard. And it also probably explains why you’re here – you’re in the desert.

Beyond just providing this different point of view, the aim of the Concrete Practice section – which I pointed more desperate readers to earlier – is to provide some solid practical suggestions as to what you can do to deliver your project successfully, even when it feels like it might be impossible.

[Still needs work – doesn’t flow into the next section]

### Project management for software development is hard

The focus of this book is on delivering software development projects, but a lot of the ideas might apply to the management of other kinds of projects. I’m loathe to make any stronger claims than that because I’m aware of how my temples pulse when it gets explained to me that software development is like building a car or building a house. I don’t think software development is much like either of those things. So, I certainly don’t want to try to tell anybody who’s business is building a car, or building a house, or publishing a book, or delivering a marketing campaign, how they can do that.

There might be some stuff in here that’s useful for you.

Do we need another book on software development project management? Hasn’t that problem been solved? Well, it would seem that the whole field still needs help.

There are various statistics. The gist of them is that fewer than fifty percent of all software development projects can be counted as successful (the figure varies between thirty and forty percent).

A big proportion of non-successful projects are categorised as “challenged.” This means that they took a lot longer than initially expected or cost a lot more. Or both. The remaining fifteen to twenty percent of projects are counted as straight out failures.

These aren’t great statistics. Can you imagine any other endeavour in your life that you’d be comfortable embarking on, if the chances of success were substantially less than fifty percent? OK, I know, marriage. But do you really want to get that emotionally, physically and financially involved?

[This is a bit weak, it needs re-thinking]

### Project management is stressful

Even if you don’t totally marry your project, the large amounts of money involved, and the relatively low chances of success, combine with other factors to make managing software development a stressful experience.

This isn’t just for the people managing the development of software, it’s often also stressful for the people who commission the software and pay for its production.

Part of what makes the business of commissioning software so stressful is that more and more often the requirement for the software being commissioned is non-negotiable. As Marc Andreesson pointed out in 2011. Software is eating the world. In most businesses the cost and complexity of the software involved continues to increase with no sign of relenting.

But at the same time, in many businesses, from music production, to food delivery, to training provision, the cost of not doing business with software is – not doing business. In many businesses, to do business means to do it with software.

Often, ultimately, massive, pressured demand for software falls on the people who write it. And that can be unpleasant. A complaint that I’ve had to deal with more than once from clients who have commissioned software is that the developers who are writing the software don’t look stressed enough.

I sometimes wonder if every morning before stand-up I lead a New Zealand Rugby-style haka, they might think the team are taking the project more “seriously.”

### Project management is important

There’s a lot of software development going on. That means there’s a lot of project management needs doing.

A company that can get good at project management could have a big advantage over its competitors. Why? Because good project management culture appears to be extremely rare, difficult to develop in-house, and very difficult to transfer from one organization to another.

### Project management is full of people telling you it’s easy

At the same time, there are lot of people trying to create the impression that good project management is easy. All you have to do is pay the fees and attend their course!

I’ve written and run training courses myself. Taking time away from the office to think about these issues is a good idea. Trying to talk about a common approach, using common terms is a good idea. But there is a serious limit to what can be done in a one, two, or even five-day course.

[Some other points here – the problem isn’t solved by pretending it’s simple]

### Project management isn’t going to be eaten by software any time soon

Software is consuming a lot of things. It’s consuming driving. It’s consuming shopping, dating and sex. With the advent of devops and cloud-based server services like AWS, software is consuming \*some\* aspects of software development.

But software isn’t consuming project management. The processes involved in taking a new idea and implementing it is are still very human. They require seeing and understanding and acting on what has been seen and understood. It’s challenging, to put it mildly.

[This is categorising what project management is – there needs to be more here]

Also, as software becomes more and more important to everybody’s lives, more and more people are discovering that they work in technology organizations. A lot of people are also in major denial about working in technology organizations.

[So what? What’s the point?]

### Successful project management is about seeing contradictions, acknowledging and managing contradictions

Most of what I’ve said in this introduction is uncontroversial. All you need to concede - to concede that project management is important - is that projects are important, and also that project management can have any effect at all on their likelihood of success.

All you need to concede to concede that project management is hard is to look at the number of projects that “fail.”

[Get rid of this conceding stuff – need a different transitioning paragraph]

There’s perhaps more contention there because there’s an entire training and certification industry which implies, if not directly stating, that project management isn’t that difficult so long as you pay for the courses and follow a prescribed methodology.

This final point is a bit more contentious, and it’s what the rest of the book, in one way or another is going to be about.

There is a view of project management that says that it should be simple. Project management is doing what you say you’re going to do. Delivering on time and budget is keeping your promises.

What if I told you that successful project management isn’t doing what you say you’re going to do? What if I told you that successful project management is about trying to do a bit of what you’re saying you’re going to do, seeing if that’s possible and seeing how the world reacts? What if I told you that project management is about finding out what needs doing and doing that? What if I told you that real project management is about trashing the dream of a project whilst, at the same time, building a reality that you can live with? Well, if I were telling you these things, it would be a whole lot like you were reading this book.

## Concepts

### The virtual value stream vs the virtuous value stream

"I shall either find a way or make one."

A value stream is a series of process steps that a “thing” goes through, each of which adds value.

One big branch of Agile project management thinking – Lean - has nothing to do with software development at all, certainly in its origins. Lean thinking has its origins in manufacturing.

To quote from Wikipedia:

“The value stream is depicted as an end-to-end collection of value-adding activities that create an overall result for a customer, stakeholder, or end-user.”

One of the central ideas in Lean is waste. In the manufacture of real, physical, things, waste is any activity which doesn’t result in value being added to the product. That includes time left waiting around between value adding steps.

In many ways, Agile methods for software development can be seen as an attempt to take Lean ways of thinking and use them in a non-manufacturing, software development environment. But how good an idea is that? How good a fit is it?

One aspect of this that might not be regarded as such a good fit is the concept of the value stream. Some of the great, inspirational books about Lean are written about the manufacture of things for which the value stream is well understood. Shingeo Shingo writes about the manufacture of washing machines. Taiichi Ohno writes about the manufacture of cars. Both cars and washing machines have value streams that are understood – now. But it took decades to get to that point (in the case of Toyota, now, getting on for nearly a century).

Reading, especially Taiichi Ohno’s book about the experience of Toyota in the manufacture of cars is that over seventy, eighty, ninety years, the Toyota company evolved two things. Firstly, what a car was (it’s hard maybe for us to imagine now, but in the thirties, this wasn’t as clear as maybe we think it is today). Secondly what the problem was that needed solving if you were in the business of making cars.

The answer that Toyota came up with was a very different answer from the answer that the Americans came up with. Toyota’s answer, which evolved over nearly a century, was to make cars at the rate of need.

If you make cars at the rate that they’re needed, then the way that you make cars is very different.

You don’t store as many parts. But you get very good at ordering the parts that you need and making sure that they arrive when you need them. And this change in the way that you order parts changes the way that you treat and relate to you suppliers.

You get very interested in what your customers want and when they want it and that changes your relationships with your customers.

You get very focused on avoiding errors and that changes the way you treat your employees and the way that you respond to ideas for improvement of the way that things are made.

Making cars at the rate of need means first making one kind of car and then making another kind of car, with the same people in the same factory.

When talking about Toyota and the system that they evolved, demand from the customer is often referred to as “pull.” With the use of this word comes an image of perfectly formed cars being “pulled” from the production line at exactly the rate that they’re wanted With parts arriving at the side of the product line, ready to be put onto the cars, ready to “add value”, triggered by an order from the customer, “Just in time” at the rate that they’re needed.

People who develop software and people who manage the development of software have taken notice of the success of Toyota and other Japanese companies that have used similar approaches.

What if waste in software development were the production of anything for which there wasn’t pull?

#### Little Acorns

Take the example of a cake. Let’s forget about where the eggs, sugar, butter and flour come from for a moment. Imagine that those ingredients are the starting point for a cake. Combining those ingredients into a batter is a value adding step. Putting that batter into a cake tin is a value-adding step. Putting the cake in a heated oven (for the right amount of time) is a further value-adding step. Decorating the baked cake is a value-adding step.

If a project has started there is some evidence that the organization can provide the dream with value. This is the virtual value stream.

Somebody has had an idea, they have managed to present that in such a way that it’s attracted funding. And that funding has been released to get the realization of the project started.

At this stage there’s no evidence that the organization can get any value out of the reality. This is the virtuous value stream.

But this is the thing that I’ve recently come to realise. Starting to make the idea into a reality is threatening to the idea.

How? OK, let’s say that Mr Customer has an idea for a cake made with a novel flour – flour from edible Mediterranean acorns (safety warning, edible acorns exists, but only in countries around the Mediterranean, the ones I would find in a park in England aren’t edible). Mr Customer’s idea is that we could make a chocolate cake with this flour and it would be gluten free. Also, all acorn flour is organic because it’s not really grown commercially yet. Let’s say that Mr Customer takes his idea to the board of Alternative Cakes Inc. They like the idea. They greenlight the project. It’s Christmas and Mr Customer promises them a delicious gluten-free organic acorn cake for Easter.

Ms Supplier is the baker. She’s tasked with baking this cake. She tries to get a hold of Mr C to see if he has a recipe for acorn-flour cake. He says he’ll get back to her. He doesn’t. His success with the acorn flour cake bid means that he’s now in demand all over the company working on other leading-edge baking projects. He’s busy on a parsnip flour cupcake and doesn’t really have time to talk.

So, Ms S tries to get on without Mr C. She tries to source her own acorn flour. She can’t find any acorn flour that is less than 80% wheat flour. She contacts Mr C and asks if he knows of a supplier of pure acorn flour.

Mr C is now pretty short with Ms S.

“You’re the baker,” he says, “it’s nearly Easter. I have to present this acorn cake to the board in a week’s time and then I’ve promised three hundred cakes for the Nuneaton squirrel festival two weeks later. I need you to hurry up and get me the cakes. I saw acorn flour on the Greek island of Kea. I thought you were a *professional* baker. Maybe we made a mistake when we hired you.”

Ms S feels slighted, and also worried that she might lose the contracts. She spends a couple of weeks trying to contact bakers in Kea to no avail. Finally, she gets on a plane and gooe out there. She follows her nose from the port and walk into Adonis’ bakery. She sees acorn flour products in a special display, gets a bit excited. She can tell from the smell that Adonis is a good baker.

She asks him about the acorn products. He shrugs. It’s interesting as a novelty. But any product that uses acorn flour needs to be at least eighty percent wheat flour. Why? Because the acorns have a powerful savoury taste – a bit like malted burnt toast. A hundred percent acorn flour cake is inedible. The other thing is that beyond twenty percent, you can’t get a cake to rise. A hundred percent acorn flour cake has the consistency of a doormat. Antonis waves in the direction of the display stand. All of these “cakes” are some variation on a brownie. The tourists buy them occasionally, but then they taste his baclava or his bougatsa or his tiropita. He smiles, “after that,” he says “they don’t want to eat the acorn cakes.”

Antonis is a brother baker and gives Ms S a couple of sacks of the acorn flour. He tells her that the name on the flour sacks is his father-in-law and he can send her more flour whenever she wants. He’s the biggest producer of acorn flour in Kea, but really, only because there’s an EU grant making it worth his while.

Ms S pay the excess baggage fee on the acorn flour and fly it back to England. When she gets back, as a starting point, she tries to make traditional cakes with one hundred percent acorn flour. Adonis is right. It’s a miserable experience. A sponge cake made with just acorn flour is like an offensive weapon. You’d feel more comfortable threatening to hit someone with it rather than taking a bite. And if you do take a bite. Oof. Burnt. Spicy, but burnt. And the sweeter she tries to make it to try to get rid of the burnt taste, the weirder the taste of the spice becomes.

After another week of experimenting, Mr S finally has a chocolate and cherry “acorn” brownie. And using other gluten free flours, such as almond, she manages to make something that’s roughly what Mr Customer asked for – a gluten free acorn flour cake.

Mrs has’t talked to Mr C in weeks, but he seems pleased and relieved when I say that I’ll have something to take to the Nuneaton squirrel festival.

But then, when Mr C tastes the brownies at the squirrel festival, he’s furious.

“I thought we’d agreed this was going to be a cake. This is a brownie. And what’s that weird burnt taste?”

“Yes, I know that’s what we said,” says Ms S “but…”

“No buts,” says Mr C, “I want a cake.”

Now Ms S is furious. She reaches under the stall and brings out a plastic box which contains a heavy, blunt, object – it’s one hundred percent acorn flour sponge cake.

Ms S offers Mr C a piece.

“Jesus, that’s disgusting.”

“I know. I think that maybe we need to rethink…”

“I’m not rethinking anything. I promised the board an acorn cake and that’s what they’re going to get. I’ve got another board meeting in a month’s time. You’d better have an acorn cake for me to share with the board, otherwise you’re fired.”

As Mr C storms away, a small lady with dark hair in chef’s whites walks up to the stall.

“You’re using acorns?”

“Yes,” you say, dejectedly. The little woman in whites introduces herself as Marina. Then she wrinkles her nose.

“My nonna used to cook with acorns. Not so good in cakes. Best my nonna ever managed was in a plum clafouti – an Italian version of a brownie. Of course they make delicious pasta.”

“What?”

“Yes, you didn’t know?”

“I did not. You mean like spaghetti?”

“No, like ravioli mix the acorn flour half and with wheat flour”

Ms S hurriedly packs up her stall at the squirrel festival - nobody seems to want to taste her acorn brownies anyway - and get back to your development kitchen. You know that there’s a type of gluten-free rice pasta that’s been quite popular for a couple of millenia – noodles! If you can make a rice-flour based pasta flavoured with the acorns. Then just maybe….

The day before the board meeting you manage to get a few minutes of Mr C’s time.

“Where’s the cake? You’d better have the cake.” He says.

“I don’t have the cake.”

“Then you’re fired.”

“Look, I know, you hired me to make you an acorn flour-based cake. I didn’t manage that but…”

“But, but, it’s always but with you.”

“But I’ve got these…”

Again, I show him a plastic tub of heavy brown squares.

“What are these? Biscuits that we could use as anti-aircraft shells?”

“Acorn and rice flour ravioli.”

“Why are we even having this conversation? This is a cake company.”

“Please – give me ten minutes in the development kitchen and I think I can change your mind.”

You serve Mr C and the board ravioli covered in sage-flavoured olive oil and filled with a vegan ricotta. These babies are gluten and lactose free and vegan and they use organic acorn flour.

Mr C loves them. He eats them all, “that burnt spicy taste!”. As he eats he keeps complaining about how he wishes just for once he could hire the kind of decent, honourable, reliable staff you do what they say they’re going to do.

“From little acorns: how acorn pasta became the new food and health sensation” says the headline on the supermarket magazine. On the front cover is a picture of Mr C about to eat a forkful of acorn ravioli, inside is an interview where he explains his journey towards this culinary discovery. There’s no mention of sponge cake, there’s no mention of Ms S.

#### I thought we were talking about software development, what the hell has any of this got to do with acorns?

I seem to spend my life now standing in cafes (I really like going to cafes) behind someone who is asking about the lactose free coffee options and the gluten free cake possibilities. My guess is that a lot of the requests that a cake company deals with are for gluten free cakes.

I know nothing about the cake making industry beyond that experience - the standing in the queues at cafes.

The above story is an analogy for something that I do know a bit more about – software development projects.

How is making acorn cakes like delivering software projects? OK, let’s list a few of the ways.

##### A particular appealing brief has been sold to senior stakeholders

Mr C had an idea and, somehow, he managed to persuade the board to fund that idea. Like most (relatively) sane and (relatively) honest human beings. He believes that you should do what you say you’re going to do. Like most (relatively) sane and (relatively) honest human beings, he wants to avoid a conversation with his bosses where has to explain why he has taken their money and not done what he said was going to do with that money.

##### What’s being asked for is impossible

The way that Mr C constructed an argument that appealed to the board for his acorn cake was really a substitution argument.

* We are constantly being asked for gluten free cakes
* Cakes are made from flour
* Acorns are gluten free
* Acorns can be made into flour
* THEREFORE: We can make gluten-free acorn flour cakes.

Mr C didn’t check his idea with reality. In one sense, why should he? He had an idea, an argument that looked good on paper, that he could make acorn cakes. He managed to persuade the board to fund his idea. And really they didn’t need much persuading, because they already knew (because everybody was asking them) that there was a need for a gluten free cake.

##### Being told that what’s being asked for is impossible, results in panic, threats, denial and accusations from Mr Customer

This is the crux of this book. Attempting to deliver a dream threatens the dream. By attempting to make gluten free acorn cakes and encountering problems, Ms S starts to attack the idea that “We can make gluten-free acorn cakes” is even possible. But also, Ms S undermines something else that is held very dear to Mr C – the idea that he should keep his promises.

Aside: What we’ve run up against here is a fundamental principle of human nature, it’s what the psychologist Robert Cialdini calls “commitment and consistency” and it’s a powerful force. It’s so powerful and so pervasive that often we hardly notice that it’s there. The principle of commitment and consistency is this:

We should do what we say we are going to do

This causes a lot of trouble in project management.

##### Understanding the real possibilities of the problem space results in a solution

In our little acorn story, the saviour was a little Italian lady. But what if Ms S had decided that she didn’t want to rely on luck. If Ms S had decided that she didn’t want to rely on luck she would have had a group of people on whom she’d try out all of her kitchen creations.

She wouldn’t have relied on her own taste buds only to tell her that the acorn sponge was inedible. She would have had a group of tasters who would have told her. Likewise, when she cooked acorn and rice ravioli she would have had a group of tasters to tell her it was delicious, how much they’d pay for it, what sorts of fillings they would like.

She would have collected together a group of people representative of the kind of people who want gluten free cakes.

She might also have made contact with people inside the company beyond Mr Customer.

##### Neither the board nor Mr Customer are happy with the solution

Of course, in one sense, the success of acorn ravioli is, well, a success. In another sense, Mr Customer’s idea for a gluten-free replacement for the sponge cake just didn’t happen. It’s perfectly possible that despite the success of the acorn ravioli, Mr C continues to be mad at Ms S that she didn’t deliver the acorn sponge cake.

Here’s an even weirder thing – it’s perfectly possible that Ms S continues to blame herself for not delivering the acorn sponge cake. Such is the power of dreams.

Of course, if the acorn ravioli is a success, then Mr C will have a whole host of other problems. Remember, acorn flour isn’t really commercially produced. Oak tree plantations take about thirty years to come online. Maybe there’s a solution to this problem, maybe there isn’t.

##### Real consumers are happy with the heavily modified solution

The product that emerged from this project – acorn and rice flour ravioli – looked nothing like the product that the project imagined – an acorn flour sponge cake. But the product that emerged was a success, it delivered value to its customers and revenue (we’re not sure yet whether there’s actually any profit).

#### The Perfect Project Management Machine

In his book “The Innovation Algorithm,” Genrich Altshuller suggests an important question to ask when trying to invent a solution to an identified problem.

**What would be the perfect machine?**

You might imagine that the perfect software development project management machine would be like this:

You put requirements in one end, and then software comes out the other end. And this would indeed be perfect, if we added to the fantasy that the machine could turn requirements into working software in no time and at no cost. If implementing the requirements does involve some time and some cost, we might imagine that it’s a good idea to get all the requirements right before we put them through the machine. This also sounds like a good idea, again, until we realise that writing the requirements also takes time and money. And, if we’ve ever heard a story like the one above about the acorn flour – where the initial requirements didn’t work.

What if a perfect machine were one that could start out with rough requirements – not much more than an area to explore – the software equivalent “do something interesting and tasty with acorn flour”. What if the ideal final result was a piece of software that delivered value to its customers and revenue to its owners and did that by delivering the functions that they wanted and needed, even if they weren’t initially imagined by those that commissioned the project – the software equivalent of rice and acorn ravioli?

This kind of perfect machine of project management, not only does what it’s told, it discovers what it needs to do. You point it at a problem area and it identifies value and produces software that provides that value in that area.

But what if we extend the fantasy of the perfect project management machine? What if the perfect project management machine is one that not only takes a requirement and uses it to create something of value to users and its owners? What if the perfect project management machine also does this in a way which is valuable and rewarding to the person who came up with the initial idea?

Hey, let’s go completely crazy and say that a perfect project management machine would be one that left everybody involved in the process better off. The customer who asked for the software, the people who use the software and the people who built the software. Wouldn’t that be great?

### Some vs all – small batches

The idea of a perfect machine is an interesting one. Yes, of course, a perfect machine would be one that did exactly what you asked of it in no time and at no cost. But let’s imagine for a moment a slightly less perfect machine. One that does exactly what you ask it in no time, but it costs the same as a normal project management process.

This machine can write your software for you in an instant, but it still costs the same as it would if it took six months.

How would you use such a machine? Well, one temptation would be to try to get all the requirements that you fed to the machine perfect before you put them in.

But making sure they were perfect before they got fed to the machine might take a long time.

That might take so much time that the machine taking no time at all stops being that much of an advantage.

What about feeding instructions to the machine in small chunks?

But. This is crucial. When you get a bit that works, you then show it to everybody. By everybody I mean, the people who might use it and the people who might pay for it.

OK. So here’s a question. On your project, the project that made you pick up this book, which bit would you feed through the machine first? Which bit would you like to see working first?

And now I’ve got another question. Did your brow furrow as you started to answer this question? Do you know why your brow furrowed? Because this question – which bit should we do first - is a difficult question and there isn’t just one answer. It involves thinking.

“On time and to budget.”

“Totally secure.”

“Completely specified.”

“We need everything on the list.”

What do all these phrases have in common?

They don’t require any thinking.

If something is going to be late and over-budget and/or over-budget, we need to do lots of thinking. We might need to say how late, we might need to agree how much over budget. We may need to have a lot of difficult conversations with angry and disappointed people.

If something is “totally secure” we don’t have to worry about it. If something is “partially secure”. We need to continue to worry, we have to quantify the ways in which it isn’t secure.

If something is completely specified, we don’t have to worry that we’ve got the level of detail correct in any particular part of the specification. We don’t acknowledge the reality – that we’ll have got some of those specifications wrong. Some of them won’t be sufficiently detailed. Some of them will be too detailed. Some of them will just be plain wrong. Some of them will be self-contradictory.

Nobody wants to think much. But it’s really rude to tell them that.

People who try to help people remove themselves from cults have a term for these kinds of phrases: “thought terminating.” Interestingly, people who are trying to rescue people from a cult aren’t aiming to get the people who are victims of a cult to change their mind. Rather, what they’re trying to do is to get those people to think for themselves.

Here, right in the middle of this section we might have stumbled upon the central game that is going on between project managers and the people who want projects. And the game is about trying to get them to think. Actually, the game is about tricking them into thinking. Because people really don’t want to think, and so if you ask them to think they are very good at batting away the attempts with such phrases as “just do what I say” and “we need everything by the 15th, no argument.”

We’re shaped by evolution to do as little thinking as we can get away with. The reason for this is that thinking is very energy consuming. The trouble of course happens when the amount of thinking that we can get away with is less than the amount required for survival. And getting that level right requires – well, thinking.

In project management, everybody needs to do a bit more thinking and that includes the project manager as well as the client. It’s not enough to realise that the project is impossible, you h

ave to start to think about how you might get the client to think that it’s impossible. And baldly stating the facts of how bad things are isn’t all that sophisticated.

I’ve done this repeatedly, and it’s gone badly. What I’ve come to realise is that one of the fundamental jobs of the project manager is to be clear what this bad news is and when required get other people to think about it. There is similarity here I think between someone who’s stuck in a cult and someone who’s stuck in the virtual value stream.

Staying with the original idea for a project is comfortable. Staying on the compound with the guru, is, in some sense at least, comfortable.

Leaving the compound has an exit cost. But outside the compound are all the benefits, and dangers, of the real world you have to keep your wits about you and you have to use them to think.

Talking of thinking. Are you thinking what I’m thinking? “Delivering the Impossible” might be a thought-terminating phrase. Because delivering the possible needs a lot more thought. What is possible? What needs to be done to make that possible? What would be the highest value thing to do first? What would create pull?

You could ask other people for an answer.

You could ask the users. “We’re thinking of writing some software that works in this subject area. What’s the first things that it should do?”

You could ask people who work in the client organization, “We’re thinking of writing software that helps these customers, do this, what’s the first thing it should do.”

You could ask senior people in your client organization, “We’re thinking of writing software that helps in this area, what’s the first thing that it should do.”

You could even ask the technical team that are going to do the development.

Some of these groups might not agree with what’s the best bit to start with. Some of these groups might say “all of it, we want all of it.”

But at least then you have an idea of where to start.

### Agreed activity vs steady progress – gently tough

Imagine you are on a stage. The stage is blank. There are no props, no scenery. But, out there, beyond the lights, there is an audience. You guess that they want you to say something, but you really have no idea what they want you to say.

This is literally many people’s worst nightmare. Yes, I know if this were a real nightmare, but for the purposes of this thought experiment, everyone is clothed.

OK, imagine, that the nightmare’s not quite as bad as you thought. There’s someone else on the stage, in fact there are several people on the stage. The audience is still watching. What would you do? What would you do together?

Keith Johnstone invented his own approach to theatrical improvisation. He’s written two books about it. He has spent most of his career watching people in exactly the situation that we just imagine. Actors who are trying to improvise a scene are in this situation.

He noticed a few things.

It doesn’t matter that the actors chose to be there. The fear is just as real.

In this “nightmare” situation, people try to do two things at the same time.

1. Stay safe
2. Do what everybody else is doing

The result is what Johnstone calls “agreed activity.”

For example, if the group on stage are told that they are in a pirate ship, rather than noticing a navy ship on the horizon sailing towards them, they will notice that the deck is dirty and needs cleaning.

One of Johnstone’s roles as a director, or instructor of improvisation, is to help move the action forward by shouting something at the actors like “the deck’s clean! You notice something on the horizon.”

Agreed activity is a risk in improvised theatre. If it happens, the result is boredom for the audience.

But agreed activity is also a risk in project management. Because the teams that work on projects, when they are expected to perform in uncertain circumstances do the same things that improvisers do on stage. They try to stay safe and they do what everybody else does.

In software development there are two main agreed activities.

1. Gathering requirements without trying to make them into working software.
2. Developing software without deploying and testing it with users.

### Trench warfare – an irresistible force vs immovable object

### The wreck of an idea – the dream vs the reality

### Emotional management – what we feel vs what we say

## Concrete Practice

### ### What do I mean by Agile exactly?

Agile for me is Scrum combined with some extreme programming practices and a Lean sensibility. Does that make it any clearer? Yeah, I thought it might not. So let’s go through those words one at a time.

#### #### Scrum

Scrum is a project management framework. It claims that it can be used for managing all sorts of projects, but it reality it’s mostly used for managing software development projects.

The way that I tend to explain Scrum is in terms of its ceremonies, roles and artefacts.

##### #### Scrum Ceremonies – Stand up

One of the distinguishing features of Scrum is that teams that are using it have a “stand-up” meeting every working day. This is a short meeting that happens every day at the same time and in the same place. Everybody who attends stands up. Each member of the team says what they did on the previous working day, what they’re going to do on the upcoming working day and mentions anything that’s blocking them.

##### Scrum Ceremonies – Planning

Planning is a meeting where the team gets together and decides what it’s going to do in the next sprint. Scrum works in sprints. A sprint is a short period of time, relative to the length of the project. And the team plans, in detail, only for that time. The length of a sprint is typically two weeks. Some times, it’s longer. Sometimes it can be shorter, for most of the teams that I’ve ever worked with, it’s two weeks and planning happens once per sprint.

##### Scrum Ceremonies – Show and Tell

Show and Tell – (it is also referred to by a bunch of other names – demo/review/showcase) is an opportunity for the team to show the product owner and also other stakeholders and, really, anyone who’s interested, what they’ve done in the past two weeks. In theory, the only thing that’s supposed to be demonstrated at Show and Tell is working software. But in recent projects that I’ve worked on, the team have show designs for software that is going to be built and also talked about the findings from user research that they’ve done as well as demonstrating working software and that’s worked really well.

##### Scrum Ceremonies – Retrospective

In the retrospective the team gets together at the end of the sprint and discusses how the sprint went. What went well, what didn’t go so well, what could be done better, what ideas they might try. It’s also an opportunity to vent – anything that’s frustrating the team – it should be able to come out in the retrospective. It’s also an opportunity for the team to identify things that might be slowing the project down, or might be killing it stone dead.

##### Scrum Roles – Scrum Master

The role of the Scrum master is to make sure that the ceremonies that we’ve talked about actually happen – that means booking rooms, encouraging people to turn up to meetings that are supposed to be there and discouraging people from turning up to meetings who aren’t supposed to be there.

The other supposed role of the scrum master is to remove the obstacles, impediments, blockers that are identified by the team or, if these are things that the Scrum master can’t do by themselves, to highlight these blockers.

But what do you do when you’ve dealt with all of the obstacles, blockers and impediments that you can deal with? You’re just left with all the obstacles, blockers and impediments that you can’t solve. At least not straight away and not without prolonged effort and the cooperation of people far outside of the team. This is why when I think of the role of Scrum Master, I often think of a quote from Jerry Springer the Opera “I don’t solve people’s problems, I televise them.”

And this, televising of the problem leads to the final important role of the Scrum Master as the boy who points out that the emperor has no clothes. The person who points out that the project can’t be delivered as it’s currently being discussed, because it’s impossible.

##### Scrum Roles – Product Owner

The idea in Scrum is that there should be one person who is responsible for a product. They should prioritise what goes into the product and they should act as the link between the team and wider organisation that wants the software.

This is a really hard role, probably impossible role. Mainly because it requires a “Goldilocks” level of authority. It might seem like a good idea to have a really senior product owner, but anybody who’s really senior isn’t going to have the time to devote to the role. Often, to the organisation that provides the product owner, it seems like a good idea to have someone who’s quite junior, because either, that’s the only person that they have available or they on some level think it needs to be someone who will say yes to all the requests from more senior people in the organisation about what the product should do.

The story of a lot of projects is trying to shore up this position.

Scrum Roles - Team Member

The rest of the team is made up of people with a combination of skills. The idea in Scrum is that those roles aren’t defined. That’s the idea. My experience is that people really want a role. Developers want to be developers. Testers want to be testers. Designers want to be designers.

Scrum Artefacts – The story

A story is something that needs doing. In general, the focus of a story should be something that’s visible to the user and so sometimes the story is referred to as a “User Story”.

Stories should have somewhere near the right amount of detail. That means that they shouldn’t be too detailed and shouldn’t be too vague.

Scrum Artefacts – The product backlog

The product backlog is a collection of stories. For a project, it’s all the stories that we know about so far.

But operations on the product backlog can be some of the most powerful things that you can do in actually managing a project. What operations.

1. Prioritise

The best thing that you can do with a product backlog is order the items in it. Sometimes, the most crucial test of the product owner is whether they can do this. One way of making this easier for the product owner is ask them not to prioritise the whole backlog, but rather just pick one or two things that they want the team to do next. And then a couple of things that should be done after that.

2. Say “yes.”

A crucial thing about prioritising the items in your backlog is that you can start to admit that some things that are on the list won’t get done, soon, or for a long time, or maybe ever. That might sound like a negative. But the positive side to this is that it’s easier to say “yes” to adding things to the backlog.

3. Size

A joke – which I’ve seen attributed to Prince Charles – heir to the throne of the United Kingdom, goes as follows.

Q: “How do you make sure that you’re never alone? Even if you’re stranded on a desert island?”

A: “I don’t know your highness.”

Q: “Always carry with the equipment required to mix a martini.”

A: “I’m sorry sir.”

Q: “When you’re feeling very lonely on your desert island, simply start to mix your martini. And wherever you are, someone will jump out from behind a tree and start telling you that that’s not the way to mix a martini.”

Another story – which I know comes from Rick Stein, the TV chef. He says that when he’s interviewing chef’s to work in his restaurant he asks them to fry an egg. The chefs that aren’t much good look worried – fearing that there a perfect “cheffy” way to fry an egg.

The good chefs – the ones that Stein is likely to hire – shrug and say something along the lines of “I don’t know how you do it, this is how I do it.”

Sizing and estimation is the martini moment of Agile methods and this is me shrugging as I make the fried egg martini.

I try to estimate everything in the backlog – if something is really vague I get the team to give it a really big number. A hundred and forty-four is normally big enough sometimes the team might reach for a number bigger than that, but that’s normally big enough.

I use complexity points, the old school Fibonacci series rather than the “modified” Fibonacci series which has an infinity on it.

I get the product owner involved in the estimation process. It’s supposed to be the team that’s going to do the work that gets involved in the estimation process. Is the product owner doing the work? You know, I think they are, or I think there’s an argument that they are.

4. Track progress

Once you have a list of things that you’re going to do, you can start to plan to do them in the planning meeting that we discussed above and then the team can work on the things that have been decided for two weeks. And at the end of that two weeks some of those things that you planned will be done. This means that you can put together something called a burn-up chart (we’ll talk about it later in concrete steps). After a few sprints, the burn up chart will start to show how much work is in the project and how quickly you’re getting through it.

5. Draw lines

Once you’ve got a burn-up chart you can start to draw lines on it. One line is the deadline. This shows how much work will be finished by the time the project is supposed to be finished. Another line is the point at which all the work in the backlog is likely to be done given current progress.

Both of these lines are vertical lines. Lines up from the timeline. But the power of putting all the work that you know about in the backlog and prioritising it is that you are admitting that not everything will be delivered. And that means that you, as a team, including the product owner, and maybe wider in the organisation, can have a discussion about where to draw a horizontal line through the product backlog.

#### #### What do I mean by ‘Lean’ exactly

Limiting the work in progress

Small batches

Focus on adding and delivering value

A close up of a map

Description automatically generated

### Ask yourself these questions

Are you being asked to deliver the impossible? If you are, the first thing to do is to get an understanding of exactly where the project is.

I’ve put together a list of questions to ask yourself about any project that you’re working on. The answers to the questions are on a 1–10 scale. This isn’t science. It’s mostly about gut feel. The number that floats into your head when you first look at a question is probably the right number.

### Q1. (On a scale of 1 to 10) How sensible is the deadline?

If there’s any description of the deadline as “aggressive” or “ambitious”, that should push the score down. If the deadline hasn’t been in any way informed by how much work there is to do and what the team is capable of, that should push it down too.

Conversely, if there’s no deadline, then score should be really low.

### Q2. How reasonable is the list of requirements?

Add points if you think there’s a list somewhere that roughly reflects what the thing is supposed to do.

Score even lower if it’s a requirements document written by a third party that no longer works on the project.

Score higher if it’s what, in Agile, is called a product backlog, a list of items, some headings, some more details that are expected to become more detailed over the course of the project.

### Q3. How good is the representation of the client?

Add points if there’s someone you can call a “product owner” or “product manager”. Add extra points if this person is available to the team most of the time. Add even more points if there is evidence that this person is representing what the organisation paying for this software and knows what they want it to do.

Take points away if the person who’s notionally “product owner” or “product manager” is either:

1. Really senior and this is just one of 10 jobs they’re currently doing.
2. Really junior, so had no authority to really make any decisions about the product, and doesn’t really have good connections within the organisation.
3. Isn’t one person, but a committee.

### Q4. How good is user engagement?

Is there evidence that the team is considering what users want and the team is checking with users how they will respond to what the team builds? Is there engagement with users all the way through the project? That means people inside the organisation that’s paying for the software and people outside the organisation that will be using the software.

Take points away if you’re told that people inside the organisation are too busy to talk to you or give you feedback on versions of the software.

Take even more points away if you’re forbidden to talk to users.

### Q5. Technical constraints?

How easy is it to realise the project technically? Does the team have a platform for development? Is there a platform for release? Are there platforms for testing? Are there any requirements to use certain architectural stacks or certain architectures? Is this causing problems or delays?

Take points away if the team is forbidden from doing what it needs to do to procure and access test environments and like-live environments.

### Q6. Physical resources?

Do the team have all the physical resources that they need to deliver what’s required? Do they have an office that they can get into? Is it habitable? Is there a working toilet (true story)? Does the Wi-Fi work? Are there desks, chairs and computers. There’s no requirement here that the team are all physically located together, but if they aren’t then this question is about the quality of the communications infrastructure.

### Q7. Team ability?

Does the team have the right mix of skills at the right levels to do what’s required?

### Q8. Team morale?

Are the team beaten down?

### Q9. Value landscape?

Does anybody want the software that the team are developing? Score this highly if it’s certain that people do. Score it low if there’s either no indication, or the indications are extremely uncertain (e.g. it’s a completely new product that no one has thought of before). Score it even lower if there’s active antagonism to the product (e.g. the people who are going to use it have been told that most of them are going to be replaced by it).

### Q10. Management landscape?

Beyond the product owner, what is the appetite and support for the product? What do senior managers in the organisation think? Score lower if you don’t know or you are certain there is genuine antagonism. Score high if you know that this could really save some CEO’s bacon.

### Q11. Independence?

To what degree is this project independent of other projects or external factors, e.g. interfacing with other systems, licensing by an external regulatory authority?

If you’re working on a project, you could answer all of these questions, divide by 10 and then get an overall score out of 10. If you think back to projects that you’ve worked on, you could do this for one of those. I did this for four projects that I’ve worked on. It was a very interesting exercise. Then next time that I get chance when I’m running a training course, I’ll run it with everyone in the room and see what their score are.

If you just did this for a project that you’re working on, how do you feel now?

My thoughts looking at the scores from the projects that I tried are that any project that scores on average, across all 10 scores, over 5.0 probably has a reasonable chance of success and projects scoring over 6.0 are the ones that we’d all like to be working on.

Conversely, any project scoring less than a 4.0 is in serious trouble. But that’s just the averages. If any individual score is below 4, that on its own can probably bring down a project. It doesn’t matter how well your project has been implemented if nobody wants it. It doesn’t matter how much your customers love what you’ve done if someone in the organisation is going to make sure that it is never released.

### Do this about the answers

#### What do to if you score yourself low on any of these questions?

##### Question 1 – Deadline (comes together with Question 2)

###### Step 1 – Make a List

If you haven’t got a list of things that need doing on the project, get a list. It needs to be a list of big-ticket items that the system can do. For example, if the system is an airline ticketing system, one of the things that this system is going to have to do is:

**Search for flights.**

Another is:

**Take payment for tickets.**

Try to make this list from the point of view of the users. But also acknowledge that there are technical set-up tasks that aren’t visible from the point of the user for example:

**Configure test and release servers.**

Aside: “We can’t break up the requirements, we need the whole thing.”

Often the response to any attempt to come up with a list of the things that need doing on a project is “we can’t break it up into a list” or “it’s a waste of time breaking our requirements into a list” or “It just needs to do exactly what the old system did.” All of these really translate into “we don’t want to think.” And who does?

But making this list is a very gentle way of doing some thinking about what’s being asked about the project. If you have this discussion with the team and some representative of the customer (a product owner if you’re lucky) it’s almost guaranteed that you’ll come out of the meeting with a better understanding of what the project needs to do.

If you do have a specification document, you can probably pull list items out of that. It might that the headings translate into items to put on our list, it might be that there are boxes in a diagram that you put on your list. What you need to end up with is a list of things that the system needs to do and a list of things that need to be set up.

What if we’ve forgotten something?

*You almost certainly have. Don’t worry. What you’re trying to do here is to capture most of it. Spoiler alert: this won’t be the only time you look at list. Further spoiler alert: big things will get added to this list.*

But to start with, you need to end up with a list like this:

|  |
| --- |
| Customer login |
| Customer registration |
| Search flights |
| Take passenger flight details |
| Make flight booking |
| Take payment |
| Offer car hire |
| Reserve specific seat |
| Add baggage |
| Offer hotel |
| Take payment in airmiles |
| Report sales |
| Include searches of partner airlines |
| Internationalisation |
| Agent login |
| Cancel Booking |
| Set up test server |
| Set up live server |

###### *Step 2. Size the items on the list.*

This is what I’d do. If you ask someone into help you, they might do it differently.

First of all, here’s the Fibonacci series.

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144.

Why use this series? Look you don’t have to use this series. There’s a modified Fibonacci series. 1, 2, 4, 8, 16, 32, 64, 128, ∞. I’m not fond of it because I don’t like the infinity symbol. I want to get to the end of this estimation process with a number. If something is really vague, I think it’s fine for the team to give it 144, if they can’t stomach that, I’d rather give it 233 than infinity.

If you’re really struggling as a team with the idea of estimating in terms of complexity, then you can, use ideal days. This is what Henryk Nyborg recommends in his book Scrum and XP from the trenches.

In a separate meeting, on a separate day you need a big board-room style meeting room with a log wall that you can stick things on. Failing that, everything that I talk about doing on the wall, you could do flat on the table.

Go back to the list of things that need doing on the project. Write each item on the list on a separate card. Index cards is fine. Or big Post its.

Write all of the Fibonacci numbers on separate Post-its. And space them out across the wall. Then, as a team, look through all the items that you’ve written on the cards. Find the one that you think is the simplest thing to do. It doesn’t matter where in the order of things to be done it is. It just needs to be the simplest.

Put that card under the 3 or the 5, depending I suppose, how simple you think it is, is it *really* simple, or just quite simple compared to the other things on the list. Now for the rest of the cards.

Hold each card up one at a time. And discuss with the team where to put it on the board. Is it as simple as the last card? Is it more complicated than the last card? How much more complicated? Keep pushing through this. Allow, ten, twenty seconds, maybe a minute of discussion between cards and then move onto the next one. If there’s any serious disagreement in the team about where a particular card should go, almost always go with the higher complexity. If you get the “we’ve no idea” reaction from the team. Or you get the “Oh my God! That’s a nightmare” reaction from the team then you should give it a high number – push it to 144 and see if then there’s any desire to pull it back to something less.

What you’re aiming at, at the end of the session, is all the cards on the wall under a Fibonacci number. Make sure then that you take plenty of photos of the wall and make sure that you record the numbers that go with each of the items in some kind of Agile Lifecycle Management tool – Trello and Jira are the most popular.

Actually, what you’re aiming at is a number. A single total for all the work that’s in the backlog. So at the end of this meeting you have a list like this:

|  |  |
| --- | --- |
| Customer login | 34 |
| Customer registration | 34 |
| Search flights | 55 |
| Take passenger flight details | 55 |
| Make flight booking | 89 |
| Take payment | 34 |
| Offer car hire | 5 |
| Reserve specific seat | 89 |
| Add baggage | 34 |
| Offer hotel | 13 |
| Take payment in airmiles | 89 |
| Report sales | 55 |
| Include searches of partner airlines | 144 |
| Internationalisation | 55 |
| Agent login | 8 |
| Cancel Booking | 144 |
| Set up test server | 144 |
| Set up live server | 144 |
| **Total** | **1225** |

###### Step 3. Track Progress

In a separate meeting. Go through this list again. Decide which of these items is “Done” – put them in the “Done” column. Decide which of these items are currently in progress and put these in a “To Do” column. Put a percentage “Done” on them.

OK, I would be being dishonest if I didn’t own up to that last sentence being – well – controversial. Because there are several different ways that you could decide on the “percentage” done. For each of these items.

One very good way, if you have a “mature, well-groomed” backlog of stories collected into epics is have epics as the items in this list and then only count as “Done” the stories that are “Done” inside the epic.

Another way is to let the team “roughly” estimate the percentage done of any item. This will seem outrageous to some people. All I can say is that I’ve made it work.

What’s important is that you don’t just do this once. You keep doing it. Every sprint, every fortnight. What’s also important is that if you’re going to be doing it that often, you keep it short. Half an hour is ideal. Do this for three weeks, you’re going to keep doing it all the way through the project, but for now, do it for three weeks. Then you can move onto the next step.

###### Step 4. Put it All together

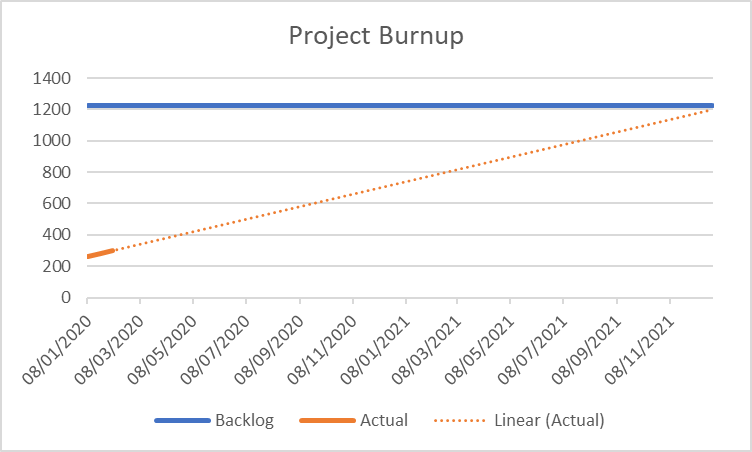
OK. Now you have the ingredients to start challenging, fixing, or setting the deadline for your project. He’s what you’ve got.

1. A total number of points for all the items that need doing on the project. (You got this from the meeting in step 1 and the meeting in step 2).
2. A percentage through each of those items from three different meetings spaced two weeks apart.

What this means is that you end up with a table that looks like this.



###### And from this table – from this sparse amount of data, you can get this.



This is a burnup chart. It shows all the work that we know about in the project (the blue line) and all the work that we’ve done in the project (the orange line). It also shows where those two lines meet.

Here’s something that you should know. If you do this exercise, the news that it delivers will almost always be bad. It will almost always be jaw-droppingly, embarrassingly bad.

The worst that I’ve ever seen as a result of this exercise is that the burn-up graph showed that the work was going to take, at the current rate of progress SEVEN TIMES as long as it needed to to meet a hard, non-negotiable government mandated deadline. The best that I’ve ever seen is that it came in at just one and a half times as long as required. I’ve never seen it come in under.

Let’s just take a moment to think about why, in ten years of managing software delivery, I’ve never seen a project that looks like it will deliver sooner than expected.

Here’s what I think. I think it’s the nature of the virtual value stream. For a project to get funding it has to promise a lot, it has to overstate its value. At the same time, it has to understate its costs, both financial and psychological. Psychological costs? Yes – changes in ways of working and thinking that will result from using the outcome of the project.

Another way of thinking about this is that if the project was honest about the uncertainty of its value and the size of the costs, it probably wouldn’t look attractive enough to get funded. Yes, that’s right. One of the hidden/accidental criteria for a project to get green lit is that it has a concealed burnup chart like the one you’ve discovered.

Are you getting vertigo? Sit down, deep breaths. Make up a favourite beverage of your choice. If that’s something stronger than peppermint tea, I’m not going to judge. But while you’re sipping. Maybe you could reflect in this. You picked up a book called “Delivering the Impossible.” Why did you do that? Probably because you suspected, or *thought* that you *knew,* that a project that you’re working on, or connected to, is impossible, as it’s currently framed.

What I’ve done is walk you a process of finding out for certain. I know it doesn’t feel like it now. But this is progress. You were right. The project is impossible. Now what are you going to do about it?

OK, there’s one option that I’m going to talk about as if it’s the only real option. But I think it’s important to talk about at least several others.

Walk away

How bought in to this project are you? How embedded are you? If you can simply not get involved in this project and move onto another and the deadlines and timescales are particularly unrealistic, you might want to move onto something else. This might be a particularly good idea if the scores for some of the other questions that I asked above are low. Some projects are almost only that in name only, they’re doomed to fail for a number of reasons, they can’t be saved. If you don’t want to work on them. Don’t.

Stop the project

Are you the person who commissioned this project? Is this project being led or managed by someone who works for you? Are you responsible for the funds for this project? Will it be your head if the project makes headlines inside the organisation, or even actual headlines in the newspapers for all the wrong reasons.

If you do have the power to stop the project, you might want to do that.

Do Nothing

Here’s an old joke. A naïve young man starts a job as a signalman for the railway. He’s done his training, but this is his first day on the job. The old signalman tests him on his knowledge.

Signalman: What would you do if the down train is coming down the down line?

Young Man: I’d pull this lever here.

OS: Very good, what would you do if the up train was coming up the up line and the branch line train was in the siding?

YM: I’d pull this lever here. I’d flick this switch here and I’d ring this bell.

And so on. The questions get more and more complicated. Until finally a question involving up trains on downlines, down trains on uplines, special excursions, royal trains, a train of nuclear waste with an escort and a herd of cattle gets to much for the young man. And he answers.

YM: I’d ring my cousin.

OS: What? Why?

YM: I’d want him to get up here. He’s never seen a train crash before.

Here’s the weird thing. I almost hesitated to include this option because it sounds so cynical and negative. But here’s the even weirder thing about that.

**It’s the most common option.**

That’s right. Understanding that a project is going to be a disaster, but doing nothing about it, just simply watching the train crash, is the most common option. Why? Well there are a bunch of reasons. We’re going to see one of them in the next step when we try to deliver the bad news. Nobody wants to hear the bad news. People really do try to kill (give bad performance reviews, fire, demote) the messenger.

There are some other reasons that I talk about in detail in the concepts section of this book as agreed activity. Agreed activity combines two strong human urges. One to keep doing what you’ve always done and two to do what everybody else is doing. There might be even be a third powerful human urge combined here. Stay safe. All of these things are combined in the wonderful phrase “don’t rock the boat.”

Unfortunately, by doing the thinking that we’ve done above about requirements and deadline, we’ve figured out that the boat is heading over a waterfall. It needs to change course. The people in the boat need to stop rowing in the direction that they’re heading. Not rocking the boat is a very bad idea indeed. It needs to be fucking rocked.

But just because not rocking the boat is a bad idea, that doesn’t mean that rocking the boat is going to be easy. Rocking the boat is going to be exactly like it sounds. A rough ride.

###### Step 5. Deliver the bad news

If you suspected that your project was impossible, once you’ve gone through the steps that we talk about above, you will very like be looking at some bad news which confirms your suspicions. You will have in your possession some bad news. What to do with it now? Well, you need to make that news available to people who can do something differently to allow the project to be delivered.

I would be lying if I tried to tell you that I knew a perfect answer to this question of how to deliver the bad news. Part of the answer I think is to bring them along in the process. Certainly the product owner or product manager should be involved in the discussions that lead up to creating the burn up chart.

It’s also important to have a regular meeting where delivering the project is discussed with the people who got the project funded. This might be called a risks and issues meeting. If it is, one of the risks should be, bluntly, that the project can’t deliver on time. I predict that when you try to add that as a risk you get resistance, possibly of the “It’s just unthinkable that this project can’t deliver on time, it has to.” If that’s the case, point out that everything else that’s being done on the project is to make sure that that risk doesn’t become an eventuality. But make sure that it stays on the list of risks and gets discussed.

You’re a bad person

The easiest thing to do when faced with bad news is to deny it. And blaming the messenger is one of the easiest forms of denial. What’s important to notice when this happens is that there was no discussion of you being a bad person until the bad news emerged. Are the senior stakeholders and customers really going to say that you’re a bad person? Well, they probably won’t say it in exactly those words. It will probably be wrapped in some kind of, well, in some kind of mindfuck.

“I thought when we hired you, you were the kind of person who will be able to deliver this kind of project”

“Are you saying that your company hasn't hired the right people?”

Your team are bad people – they should be destroying their lives to make sure my project is delivered on time

This often comes together with a complaint that your team aren't working hard enough; that they aren't working long enough hours; and that they aren't looking stressed enough. I must admit that this is one of the hardest objections for me to deal with without getting angry. No forget that I am going to get angry, but what's important is that I don't show my anger to the client. What's also important is that I realise the reason that the clients saying this. The client is in denial.

The team will get faster

This might true. They won’t triple in speed. They won't get *seven times* faster. But this is an opportunity to admit that the client has a point. Jumping ahead to some of the other questions regarding technical environment this might also be an opportunity to mention any regulatory or technical or physical constraints that are slowing the team down.

Complexity points? Burnups? I told you this Agile doesn’t work

Now they're just being insulting. One way of seeing all these responses is as an attempt to start an argument that detracts from the reality of the project's position.

Aside: Sometimes when I'm thinking about this stage of a project: when I'm delivering bad news, I think of a scene from a movie. it's no particular movie it's a scene in lots of movies where someone is being chased by a terrible monster. The monster is chasing them through the house they run up the stairs and they shut their door they pushed the chest of drawers or the wardrobe in front of the door. And then the monster starts to break Through the wardrobe at this stage the person who's been chased starts to throw anything that they can find at the doorway anything that might have a chance no matter how feeble of stopping the monster.

None of my other teams are telling me anything like this

Oh really? My guess is that some of them are. And even if they’re not, what difference does that make? This project isn't their project this team isn't their team.

I think we need to be positive

We are being positive. We’re really focussing on what needs to be done to deliver this project on time.

If you add five, multiply by three, take away the number first thought of and use an semi-log scale

I’m not saying that the numbers are perfect. And we’ll keep an eye on them regularly. But what the numbers are saying is that if we carry on like this we’re going to miss, not by a bit, by a lot.

I’m going to bring in an expert to tell you that you’re wrong

Actually, there’s an even better version of this:

I found this crazy person shouting at passing cars on the street and asked him about this project – he says it’s only going to take a few weeks.

I’m going to bring in someone who talks to you then talks to me so that I never hear this

You’re fired

This can happen. What's far more likely is a very veiled threat of being fired unless you stop pointing out the harsh realities regarding the project’s time scales. I'm going to be frank with you, in my ten- year career of delivering software projects, I've been kicked off two.

Both times when I was kicked off the project, I'd been there only a few weeks and hadn't managed to build up any kind of rapport. Both projects were in a terrible, terrible state.

At this crucial stage in several other projects I think there might have been a discussion between senior managers about weather I knew what I was talking about or I was just being difficult.

It's your job, it's your career, it's your mortgage. I don't remember at any point saying this was going to be easy. In fact, when you picked up a book with the title “Delivering the Impossible,” did you have any thought that it might be easy?

###### Step 6. The good news about delivering the bad news

Oh man. This is getting depressing. Talking about getting fired. But there is some good news. If you get manage to get across the bad news.

If you can manage to get across the idea that the requirements and the deadline aren’t pointing to each other and then if you can manage to keep communicating it through the project, you’ve done some very important things.

You’ve put the project on a basis of reality.

You’ve saved yourself from a world of pain, misery and recrimination later down the line when the project doesn’t deliver when and what is was supposed to deliver.

You’ve decided not to phone your cousin. What I mean is that you’ve decided that you’re not just going to keep quiet and watch the train crash like the guy in the story. You’re going to do what you possibly can to avoid the train crash. And what the joke tells us that if too many things are going on at once, there probably is going to be a crash. Using the burn-up chart and by calculating actual progress, you’ve set the scope of the project, how many things are going on, in clear focus.

Now you’ve done this, it’s worth looking at the answers to the other questions and seeing if there is anything wrong there. Once you’ve got the scope, if not under control, then at least in everybody’s sights, it’s worth worrying about the other questions.

##### Questions 3, 4, 9 and 10 – User and Client Representation and the Value Landscape

If you scored low on these questions the items that are in your backlog are probably the wrong things.

And think about it for a second, if the items that are in the backlog are the wrong things, was it even worth having the fight over deadlines?

I must admit, I’ve paid lip service to testing things with users and continuous contact with clients on some projects. But recently I worked on a project that really took talking to customers and clients seriously – before we built anything, while we were building things and after we built things. Out of a team of ten we had two user researchers. For any piece of work that we were going to do, the starting point would be talking to the people who would end up using the system. Out of that would come a list of paint points with the existing system. From that initial understanding designs were produced, and user tested, again with real users of the system, in scenarios that had been identified from the user research. If there was a problem with these designs, there might be an iteration of the designs, possibly even several times before we reached the stage where there was any agreement that the designs were ready to be developed into software.

So, there was strong work on this project to identify, not only what the users wanted and needed, but also whether the designs that we put together actually addressed what the users wanted and needed. But that was only half of the equation.

We also needed to make sure that this was what the people who were paying for the project wanted and needed – the clients, rather than the users.

In Agile, especially in Scrum, much is made of the role of “product owner”. The idea of the product owner is that they are a single person that represents the interests of the organisation regarding the product. The product owner acts as liaison between the organisation that wants the software and the organisation that is producing the software. The product owner also prioritises the order that things go into the product, which is of course crucial in light of what we know about the backlog (it’s too big) and the deadline (unless the backlog is pruned, it isn’t going to happen). But of course, if you have a fully function user research team as part of your team, the role of the product owner becomes even more complicated. Now they must, not only arbitrate between what the customers wants and what the client (the organisation that’s paying) wants. They also must prioritise and prune the list of things that the customers want.

As if that weren’t complicated enough (as if – ha!) there is also the added complication of what the larger organisation beyond the client wants.

In recent projects that I’ve worked on that have been successful, especially the one that did “full service” user research, there was also a concerted effort to talk more directly to the senior stakeholders on the project and keep them involved.

###### Do User Research, Keep Doing it, all the way through

It’s this simple. If you aren’t doing user research all the way through your project, you have no way of knowing if you’re doing the right thing, or if you’re doing the thing right.

This is research to find out who your users are. This is research to find out users’ fears, failures and frustrations with the existing system, if there is an existing system.

This is user research with customers and with clients.

I think that I used to put user research in a box. It was something that it was good to do at the beginning of a project and at the end. Now I realise that user research is the box. It’s the water that the fish of software development swims in.

What I’ve also realised is that if user research is done well it can transform a project.

###### Work with whatever product management you get

Also understand that the product owner role as it’s stated is pretty much impossible. So, do what you can to help the product owner without undermining or overruling them.

Product managers need to really understand what the product should do.

Product managers need to really understand the organisation that they’re trying to deliver a product for.

Product managers need to really understand the iterative nature of software delivery.

Product managers need to really understand the business that the product will work in.

Nobody can do all that.

###### Connect with the wider organisation

If you can connect with the wider organisation through the product owner, that’s great. But it’s also good to develop other links to the organisation.

###### Map the landing site

What’s the easiest way to find out about the landing site? Land something on it.

Aside: The project management force field. Here’s another way of thinking about a project. A possible project is the result of an equilibrium being found between a whole bunch of forces. We have what the customers want, what the value, what they respond to. We also have what the client organisation wants. But then pushing against that we have what is actually possible in the time and what is physically and technically possible. One way of looking at delivering the possible is that it’s the business of nurturing and supporting the thing in the middle of this diagram and making sure that none of these forces are too dominant. If we concentrate on what the client wants and ignore the response of customers, the project might be at least misshapen and possibly a complete bust.

As a project manager, part of your job is to make sure that all of these forces are represented and none of them dominate.

A close up of text on a white background

Description automatically generated

Figure 1. The Project Management Force Field

##### Questions 6 and 7 – Technically and Physically do you have what you need to get the job done?

You don’t have a physical location for your office? Your team don’t have computers or desks? Your team have computers and desks, but they’re not allowed in the building where the computers and the desks sitting? Your team can’t get the job done.

Office, computers, desks and team are all in place? But no internet? Your team still can’t get the job done.

The technology that you’ve been told to use is so new that it doesn’t work. The technology you’ve been told to use is so old that you can’t find anyone who’s willing to work with it. The technology that you’ve been asked to use doesn’t exist. It’s in some presentations and on some diagrams but it’s not real yet. Your team can’t get the job done.

The good news is that all these things are fixable. Offices can be hired. Security badges can be issued. Computers can be bought. Old guys can be found to program in old languages (for steep contract rates).

The bad news is… OK, there are a few pieces of bad news.

One: if have one of these problems and you can’t fix it, the project is probably impossible. Any one of these problems will kill the project.

Two: a lot of clients will think the project can still go on in spite of these problems.

Three: many clients find it insulting, irritating or embarrassing to have these problems pointed out.

Four: connecting lack of progress to the presence of these problems and doing it in a way that’s not so insulting, irritating or embarrassing that the client fires you? That’s the project manager’s job.

###### What works

Televising your problems.

“I don’t solve people’s problems, I televise them.” – the character of “Jerry Springer” in “Jerry Springer the Opera.”

Make totally sure that the client knows the problems that you’re experiencing. If you’ve been asked to write weekly reports, mention whatever is blocking you right at the top of the weekly report. Put a timer on saying how long you’ve been reporting this issue.

Sometimes writing the team’s problems on a wall works. How do I know? I’ve been told to take them down “in case someone sees them.” But also, on a project that was utterly struggling because of poor network connectivity, a wall where I invited developers to write a post-it containing the date and time of every network outage they experienced got the wifi fixed when no amount of reporting it through the usual channels had made any impact.

Negotiate a solution

Software development is a complex and expensive business.

Identifying physical issues about the environment such as Wi-Fi and office space shows just how expensive.

You need office space. It was assumed in the original contract that the client would provide offices. They haven’t. This means the client needs to pay for office space. Well, maybe. What it means is that you need to negotiate a viable solution.

The technology that you were supposed to use isn’t ready to be used in a real product. It can probably be replaced by some other technology, or, possibly it can be brought to the level of maturity required. There are a lot of options.

The database that you were supposed to be querying doesn’t exist. Maybe one could be written. Maybe this stops the project dead in its tracks.

Here’s something that you might not want to hear. Identifying issues and then managing solutions to these issues is your job as a project manager.

When we talked about product manager, we pointed out that they had about four jobs (it’s probably more) and nobody could be good at all of them. It’s the same with project manager.

A project manager should be the drummer and roadie, of the iterative process.

A project manager should be the security guard, patrolling the team.

A project manager should be the lookout.

A project manager should be the band manager – doing deals, negotiating contracts.

Nobody can do all of that. But somebody can manage all of that. When you look at that list, you’ll probably see some things that you’re good at and some that you know you’re not good at.

That’s fine.

What you need to be thinking is, which are the bits where I’m weak. How can I arrange support in those areas.

Personally, I’m not a great negotiator. When I say “not great” I might mean terrible. I just don’t have the subtlety that it requires (I think that’s what it requires, I actually have no idea). That is, I’m not going to sugar-coat it, really bad news.

But the good news is that I do know people who are good at negotiation. And these people trust me to do those other three things that a project manager is supposed to do, especially the trust me to not sugar-coat really bad news.

###### What doesn’t work

Keeping quiet

Keeping quiet about physical and technical obstacles is just another version of phoning your cousin to watch the train crash. Remember? We talked about that when we talked about deadlines. Of course, there’s a powerful natural tendency to try to make do and accommodate whatever physical and technical difficulties you encounter. Why? Because we know that alternatives are difficult.

He’s the thing. How long do you want to pretend that software development isn’t an expensive business? And who, is your pretending benefitting? Who is it harming?

In the short term, the client isn’t being troubled by the reality of how much his idea is going to cost. But in the long term, there’s just no getting away from that. In the short term you’re avoiding having an awkward discussion about the eye-watering costs of serviced offices.

Doing it in your pants

When I was at school (it was Yorkshire, it was the 70’s, have you seen Kes?), you couldn’t get off PE by claiming that you’d forgotten your kit. If PE was outside (I mean if it was football) you’d be forced to play in whatever kit you could scavenge from the lost property basket. If PE was inside, you’d be told to “do it in your pants.” Let’s leave aside the Freudian implications of that statement. What that meant was that you’d spend over an hour in the gym in just your underwear.

And again, let’s not dwell on the cruelty (I genuinely think no perversity was involved) of the PE teachers. Rather, let’s focus on the lesson that they were trying to teach.

Don’t forget your kit. For specific kinds of activities. You need specific kinds of equipment.

OK, we can’t work in the office. But we can work from home, right? Well, for a few days maybe, but beyond that, you need an office. You need a space where the team have space to meet and space to sit quietly and think and type. You need big tellies to show things. It a situation where people can point at things and discuss them.

OK we don’t need a server, we can run this stuff on our local machines. For a few days maybe, but after that you need machines that are like the ones that it will finally run on, you need machines (I know now, they’re mostly virtual) to put software on and test it.

It’s important to understand what you’re doing here and why it’s such a bad idea. By trying to make do and mend rather than do things properly, you’re instinctively not threatening the dream, the virtual value stream. But by not threatening the dream. You’re making realising the reality, the virtuous value stream, something that can deliver value in the real world, much less likely.

### Irrigate the virtuous value stream

#### Understand the potential value

#### Optimize the potential value

#### Create Pull

I haven’t read many books by someone who’s been to jail and found inspiration in them, but Gary Halbert is one.

Gary Halbert used to write junk mail. He was very good at it. He came up with an idea of a letter that offered you a book telling you the history of your surname (this was before the days of online genealogy). OK one of his letters was maybe a bit too enthusiastic. OK, it came to the attention of the federal authorities. OK he ended up in jail. That’s not what’s important right now.

What is important is that he wrote a series of newsletters on how to write marketing copy which are still available on the internet. They’re a very interesting read.

In one of the letters he talks about talking to some people who are attending one of his copy writing seminars. He asks them about an imaginary competition. Everybody in the competition is selling burgers (this was before gourmet burgers were a thing).

He wants to know from every person who is on his course, what the one thing is that they want for their burger that will that it will sell more than anybody else’s on the course.

Some people say that they want a burger made from filet steak. Some people say that they want the bun to be make of real home-made bread. Some people talk about the onions. Some people take about the cheese. Gary Halbert then says:

OK, I’ll give you \*all\* of these things for your burger.

It’s made of the best bread.

It has the best cheese.

It has the finest beef.

But if you give me this one thing, I will beat you with an ordinary burger. In fact, I will beat you with a not very good burger.

What’s this one thing?

A hungry crowd.

Something similar applies to software development. We talk all the time about the methodology (how the burger is cooked) we talk a lot about the technology and the skill of the people (what goes into the burger). But we don’t really talk much, if at all, about how much people want the software.

If people don’t want it, it really doesn’t matter how well written it is. If people don’t want it, it really doesn’t matter how perfectly the methodology was executed.

Can you make people want your software? Well, yes and no.

You probably can’t make people want something that they don’t want. But if you know what you want, you can probably make something that does that.

One of the most successful government projects that I worked on was a project to provide electronic versions of documents to a court.

The product owner for the project was a former clerk of the court. When she talked to her former colleagues, they listened to her and trusted her.

She could show them rough, early versions of a prototype. It could barely work, and they would still get excited. As she was demonstrating the software, there might be bugs. It might crash. But she could say – in a way that would never work if it weren’t one of their own who was talking “Don’t worry about that, they’ll fix that.”

And the project did have a lot of technical problems. But the main problem that it struggled with was success. Roll out of the system was supposed to be on a limited basis, in just one region. But clerks talk, once they heard good things from the clerks who were using the new system, the pressure to roll it out across the country got stronger and stronger.

Feeding a hungry crowd is a different dynamic than perfecting the perfect burger in a development kitchen.

### Focus on “some” rather than “all”

### Wreck the idea early

### Be gently tough with yourself and everyone else

### Televise the contradictions (but make it a family show)

## Conclusions

### A hungry crowd

### Maintain a smarter point of view

### Gently tough

### Celebrate success, deal profitably with failure

## Afterthoughts

### Sense and Nonsense

The title of this book is a nonsense. You can’t deliver the impossible. No one can. And I make a lot in this book about why anyone would pick up a book that has such a nonsense as a title. And I sort of make a joke of it, as if it’s an impossible question to answer. But I don’t think it’s an impossible question to answer. Question: why did you pick up a book with a nonsense in the title? Answer: because we’re being asked to deliver the impossible and reading a book about it is a way of putting off directly tacking the question in real life.

Why are we asked to swallow nonsenses?

There is a simple answer to this. Because it’s easier than thinking. We have two things that we’ve been told to believe, for example:

1. This project is essential to the future of this (multimillion pound) organisation
2. Nothing can be done about the poor state of the Wi-Fi connection.

Projects are full of nonsenses. One of the aspects of an Agile and iterative approach to delivering projects that isn’t discussed much, is that it identifies the nonsenses in a project really quickly.

We’re being asked to develop this software in six months, it’s supposed to be using a database that is predicted to deliver in two years. This is a nonsense.

We’re being asked to develop this software using a user centred design approach but we’re forbidden from talking to users. This is a nonsense.

As a project manager you’re encouraged to swallow them rather than unpack them. Why?

Unpacking nonsenses doesn’t just require thought.

Unpacking nonsenses leads… We don’t know where it leads.

For a while what I wanted to say about nonsenses was that I had this instinctive feeling that if your project was asked to live with too many nonsenses, it would probably be doomed to failure. But I wasn’t sure really what the explanation for this idea was. But I had this general feeling that if a project is predicted to be a huge success at launch, but you’ve been forbidden from talking to users, and a project is predicted to deliver on time, but that’s because it’s assumed that future progress through the work will be twice as fast as historical progress; and it’s assumed that the software will work as designed on day one after release, but the servers it will run on won’t be available until the day before. I’d assumed, just based on gut feel that “swallowing” nonsenses made your project more likely to fail.

But then I had a thought. What’s the opposite of nonsense? Sense. Something which is sensible, right and reasonable.

What’s the opposite of a nonsense?

A sense. Like the five, or six senses. An invaluable way of getting information about the world.

The advantage of swallowing nonsenses like the ones that I mention above is, for now, a quiet life.

But ha! Why is it quiet?

That’s because the disadvantage of swallowing such nonsenses is that it literally makes the project insensate. Blind, deaf and numb.

### Reading list

Robert Cialdini – Influence: The Psychology of Persuasion, Harper Business, 2006

Marc Andreesson – Why Software is Eating the World, Wall Street Journal, August 20 2011

Shigeo Shingo - Zero Quality Control: Source Inspection and the Poka-Yoke System Hardcover, Routledge, April 1986

Taiichi Ohno – The Toyota Production System, Productivity Press, 1988

Gary Halbert – The Boron Letters - CreateSpace Independent Publishing Platform, 2013

Henrik Kniberg – Scrum and XP from the Trenches, lulu.com, 2007

Keith Johnstone – Impro, Routledge, 1987

Keith Johnstone – Impro for Storytellers, Routledge, 1999