Software Project Cultures

Feeling Software Development Methodologies

Anthony Lauder

2008

Preface

It's Not My Fault!

When projects go well, it is pretty clear who we should praise: ourselves! “I did my best on that project, and it showed in the results. Without those great contributions of mine, this project wouldn't have gone anywhere near as well.”

> During my student days, I had a part time job writing software for the government. There, I came across David; one of the least capable programmers I have ever met. Whenever anything went right on a project, David was convinced it was his own doing. Whenever anything went wrong, he was certain there was a conspiracy of jealous people out to undermine him. At the end of each project, David was dismayed at the lack of a bonus to which he was surely entitled. He left in disgust. Strangely enough, projects on which he had work started to improve once he had gone. David, no doubt, was certain that his own legacy was still having a positive impact.

Who, though, can we blame for project failures? Not ourselves, of course! It's those fools over there!

> It was 1988. I was fresh out of University and had just started my first job as a computer programmer. Richard, the head of the whole company called a meeting for the software development team. "I am not happy with the number of bugs in the software. If this keeps up, I will fire all the junior programmers, and the managers will do all the programming." The new programmers - me included - sat in terror, and the more experienced ones rolled their eyes.

When a project goes bad we tend to think “Who could hope to succeed in a place like this? It was doomed from the start in this environment, with that groups of bozos on the project, and with those lazy bums responsible for a critical part of the work?”

It was 1998, I was a senior consultant with a Silicon Valley software tools vendor, heading up a team at a bank in Europe. There were several groups of contractors working at the bank, and each felt the other groups were dooming the project to failure. One group sat idle, waiting for requirements to be finalized; counting their generous per-diems, and hoping the project would never end. Another group started programming immediately on exciting technical issues; applauding one-another other for how advanced the new multi-threaded system architecture was going to be. A third group was keen to get started on writing the actual business-related software, but had little success in getting senior management approval to go ahead. With each of these groups posturing for their own perspective to dominate, the project struggled for more than a year to get anywhere.

Just about every organization has similar stories: “Those analysts are hindering projects, not helping them”; “My manager is the Pointy-Haired Boss from Dilbert.”; “That guy is a Prima Donna: too busy chasing technical perfection to ever get anything done.”

It is 2008. I am now a director at a large, well known dot com. Logan, a senior executive, couldn't figure out why deadlines kept being missed. He had been grasping at deadlines for as long as anybody remembered, and despite his continual shifting of tactics, they always escaped his grip. Logan had a new idea: "Too many projects miss their deadlines. From now on, when you hear a deadline you will subtract two weeks from it in your head. This way, all projects will be on time or early!" The managers muttered under their breath. One said to me in the corridor later: “Logan just doesn't get it; that was another dumb idea that is going to increase stress, reduce product quality, and not make a damn bit of difference to actual delivery dates”.

Best Intentions

Is the world really full of Pointy-Haired Bosses? Are projects full of incompetent team members? Do technical folks always obsess over geeky distractions at the expense of overall project success?

Sure, some folks really are going to be pretty awful, just like there are bound to be a few superstars too. On average, though, most folks aren't going to be at the extremes that war stories would have you

believe. Most managers and project team members are competent enough to to do a good job, and most have good intentions; they are trying their best to help projects turn out well.

Instinctive Beliefs

When you are pointing your finger at other people, they are pointing a finger right back at you. Those other people are just as certain that you are doing things the wrong way as you are certain they are.

The issue isn't that “we” are the smart ones and “they” are idiots. It's that different people have very different beliefs, deep down, about what makes a project succeed.

When people try to do their best, they aren't just doing what they *think* is right, or have *heard* it right, but what they *know* to be right down in their bone marrow.

Now matter how smart you are, and how great your ideas might be, you are going to have a hard time convincing other people to change their minds about how to approach their work, unless what you are saying rings true with their gut instincts.

Those instincts drive the way people approach projects willingly, and – more importantly – the way they perceive and respond to the beliefs of other people.

Methodologies

Maybe you have come across several different ways of approaching software development projects – different methodologies - each put forward as the best way to ensure project success. How do you choose the right methodology?

Maybe some folks list all the pros and cons and select the best one rationally. Maybe others jump onto the latest flavor of the month. But with tens, or even hundreds, of different methodologies out there, few people can make a rational choice, or switch constantly to whatever comes out next.

In my experience, what usually happens is that people follow a methodology that fits in with what feels right. When a methodology resonates instinctively, people will devour every detail of it, and follow it almost religiously. But if somebody is forced to use a methodology that feels wrong in their gut, they will either resist, or follow orders but feel very uncomfortable about it. It will feel like they are being forced to do something immoral; I have heard people working against their instincts say they feel like

they have sold their soul.

Metaphors and Schools of Thought

I have found, time and time again, that almost all methodologies can be grouped into one of three schools of thought. Each school of thought corresponds to one of three deeply held metaphors. The three metaphors give rise to people's gut feelings about what is right and what is plain foolhardy.

Why are there three? Obviously, these aren't the only possible metaphors, but they are by far the most common. I have come across people using other metaphors, but these usually overlap significantly with one of those presented here, or are very much a minority perspective. Rather than include a long list of eccentrics, I have gone with the mainstream.

Here, then, are the three metaphors, their corresponding schools of thought, and examples of methodologies that align well with them:

- The **Production Line** metaphor aligns well with **Waterfall** methodologies such as the Structure Systems Analysis and Design Method (SSADM)
- The **Model Building** metaphor aligns well with **Object Oriented** methodologies such as the Rational Unified Process (RUP)
- The **Customer Satisfaction** metaphor aligns well with **Agile** methodologies such as eXtreme Programming (XP)

Somebody believing deeply in one of the metaphors will likely feel very comfortable using one of the methodologies from the corresponding school of thought, and very uncomfortable using a methodology from a different school of thought. The metaphors compel us towards accepting one set of methodologies, and rejecting all others.

Let's try this out: If you have a reasonable amount of software project experience, and I say to you “Let's use a Waterfall methodology?” what kind of reaction do you have? Probably not an intellectual one. More likely, you have an instinctive reaction. Maybe a comforting feeling of security, with images forming of people working solidly, and products moving along nicely. Or you may feel a tightening in your stomach, with images of bureaucracy, and heavy-handed managers pressing down on you. These feelings don't come from intellectual analysis of a school of thought, or of a particular methodology.

They come from the instinctive pull of deeply held metaphors of software development.

Who Cares?

Well, that all sounds very interesting, but, really, what practical value is there in this?

The short answer is that an appreciation for multiple metaphors gives you a broader view on projects. Without recognizing that multiple metaphors are at play, discussions about how to approach projects often descend into dialogs for the deaf: Each person becomes so entrenched in their own position that they are unable to listen to anyone else's perspective. An appreciation for multiple metaphors helps people stand back from entrenched positions, and see more clearly their own and each other's points of view.

> I worked with a large software company with a very strict methodology, to be followed without question. More time was spent following rules than designing and writing software. Project managers were on the alert for people breaking the rules, and infractions sent ripples of blame-passing up and down the project chain.
>
> Not surprisingly, the level of bureaucracy ensured that even simple projects took months before they delivered working software into customer hands. Yet the company put complete faith in the methodology.
>
> Some project staff had long since realized that the methodology was a straight jacket – hindering projects rather than helping them. Efforts at discussing this were rejected by managers as impractical. Under-the-radar attempts to work more effectively were treated as insubordination.
>
> Project staff were left with a choice: either keep managers happy and stick with the rules no matter what, or keep customers happy and focus on delivering great results. The survival instinct meant that most projects followed the rules.
>
> The biggest problem as I saw it was that everybody knew the methodology, but very few people understood it. The project managers in particular were so focused on the letter of the law, they couldn't step back and see the spirit of it. Without the clearer perspective of the wider picture, they weren't positioned to ask whether it made sense to be working this way in the first place.

When you unravel the three software development metaphors you can:

- Better understand yourself, and why you feel comfortable working in certain ways and resist others instinctively
- See where other people are coming from, leading to mutual understanding of each other's perspectives, rather than dismissing one another as fools who just don't get it
- Realize why logic and intellectual arguments won't convert anybody to your own way of thinking, if what you are saying causes negative gut feelings in your audience
- Gain deep insights into the various schools of thoughts, rather than the superfluous distractions of individual methodologies
- Have a foundation from which to evaluate and understand new methodologies as you come across them
- Gain a wider range of perspectives with which to view a project, realizing that your own single perspective may be limiting your judgments
- Help a project team explore their own metaphors, gain insights themselves, and see new ways of working to which they may previously have been blinkered
- Act as a mediator, bridging differing perspectives, and seeking commonality and mutual respect, where previously turf battles had hindered project progress

How is this book different?

I wrote this book because people asked me to. Many of my colleagues, clients, and friends had come across various methodologies, and some had even read books on the subject. Yet they told me they still didn’t get what they were all about. They were swamped in details, and didn't know how to evaluate which was right and what was wrong. They had some gut feelings, but were nervous of taking wrong turns.

This book helps by approaching methodologies differently. Most books explain just a single methodology, or a single school of thought. Many books that do survey methodologies are swamped in detailed technical buzzwords. Others are written in such flowery language, that it sounds more like

philosophy than practical advice.

There was clearly a gap in the market. Missing was a book written in plain-words that digs into what the various schools of thought, and the methodologies within them, are really all about.

This book, though, isn't just about helping you see the broad sweep of methodologies. It is about giving you a much deeper emotional experience of them. These feelings will help you gain much deeper insights than mere words ever can. The book will guide you through each of the metaphors, and their corresponding schools of thought, and help you learn to feel in your gut what it is like to believe wholeheartedly in each of them. It is then that you will truly appreciate where other people are coming from.

Who should read this book?

This book is aimed firstly at managers, who maybe aren't swayed by technical talk or philosophical ramblings, but want to understand more deeply what various methodologies are all about. They want a fuller appreciation for which methodologies will bring them better project success. They want to know which methodologies, if any, can help them leap ahead of their competition, and which will hold them back. They want to be able to sense if a certain school of thought is just a passing fad, or whether there is something real in it for them. Above all, they want a feeling of reassurance that adopting a particular methodology isn't going to throw them into unmanageable chaos, and will actually deliver real business results. This book addresses all of those concerns, and provides a framework with which managers can develop a much deeper instinct for methodologies so they can answer these and similar concerns for themselves in the future.

The book will also be helpful to software developers, team leaders, and other technical folks who may have read some other books on methodologies, and may even have worked with several different methodologies on real projects. Even for experienced people like these, the relationships between methodologies can be confusing. This book will give such people fresh insights, that helps them understand methodologies more profoundly. It should also help them gain a broader perspective on how they and others see the world of software development, so they can seek cooperation rather than methodological turf battles.

The book may also be helpful to customers, for whom software is developed, and whose needs are, hopefully, satisfied by that software. Here they should find useful insight into how software

development organizations think, and how they, in their role as a customer fit into the various schools of thought. The bold few may even wish to nudge their software suppliers into looking at things from fresh perspectives.

Feedback

The most important part of this book is you. You and other readers will know far better than I possibly can how relevant the ideas here are to your daily work. Do let me have your feedback. I am sure I can learn far more from you, than you could ever learn from me. Together, we can make future editions of this book more valuable to its audience.

Thank you

Anthony Lauder

anthony@anthonylauder.com

Section 1: Overview

What is the best approach – the best methodology - for developing software and running software projects? In these competitive times, it helps to have any edge you can get. If somebody could point you to the best methodology, to give you that edge, you would surely follow that advice to the letter. Or would you?

I am sure that no end of people are keen to tell you what they think is the best methodology: no doubt it is the one they use themselves, or maybe even one they invented.

Chapter 1 explains why I now resist the temptation of making such recommendations. Unless a given methodology already fits in with somebody's gut feeling for what is right, they are invariably going to have a hard time accepting it. The leap of faith would almost certainly be too great for them to follow though wholeheartedly.

To really get what various methodologies are about – and have a chance of buying into them - people can't rely solely on words of recommendation and logical arguments; they need to learn to feel the methodologies instinctively.

Chapter 2 should help you find your own instinctive reactions to various methodologies, and observe which of three different methodological schools of thought feels right to you.

Chapter 3 explores how these three schools of thought mirror three basic metaphors of software development, each of which pulls human emotions in specific directions, and determines which advice somebody will accept wholeheartedly, and which they reject as impractical nonsense.

When you internalize the three metaphors, and begin to feel them and their implications in your gut, you can begin to have the same emotional experiences as people who already believe wholeheartedly in those methodologies. It is then that you will begin to understand software development methodologies – and other peoples reactions to them - at a far deeper level than is ever possible with rational analysis alone.

Chapter 1 : The Best Methodology

The Rules of the Game

Software development is a game. Like any game, it is played by rules. Well-established games, like Monopoly, Chess, and Football, have well-established rules. Software development doesn't. Since it is a relatively new game, people are still trying to work out what the rules are.

Some people claim they don't follow any rules-of-the-game when developing software. What I find this invariably means in practice is that, perhaps without being consciously aware of it, the team has learned to follows a set of conventions that work for them. Therefore, the team might not follow a published and explicit set of rules, but their conventions wrap the rules that the team is following implicitly.

> Bob Doncaster at EDP, in Milton Keynes in the UK, told me “We don't need no stinkin' rules,we just get on with projects”. In response, I listened to his team's conversations for a day. Here are some of the things I overheard:
>
> - “Nasty-boss said we have to get this done today”
> - “It's time we stopped new work, and did a massive code clean-up again”
> - “I haven't worried about performance yet; I wanted to get your input on functionality first”
> - “When are you guys coming over here to integrate your latest changes?”
>
> Even though Bob's team “just get on with the project” they clearly have plenty of conventions wrapping the rules-of-the-game they follow without question, including:
>
> - The boss may well be nasty, but we have to obey his orders
> - Periodically, freeze new work and focus all attention on cleaning up code
> - Focus on stabilizing functionality before optimizing for performance
> - Team members meet periodically to synchronize their work

The Software Crisis

In the 1950s, the rules of the software development game dictated that computer programmers wore white lab coats. You couldn't let mere mortals near computers – this was a job for scientists. Software development was all about applying mathematics, and using scientific principles to get precise and predictable results.

> PUT THE EXAMPLE OF DIJKSTRA IN THERE STILL BELIEVING IN THIS

Unfortunately, programming turned out to be a lot less scientific than people had hoped. Projects in the hands of scientists were going over budget, and taking too long to complete. Delivered software often missed required features, and quality and reliability tended to be poor.

Throughout the 1960s, the capabilities of computers increased, as did expectations of what could be done with them and the complexity of the software required to meet those expectations. As a result, the unpredictability of software projects didn't go away; it got worse.

By 1968, at the NATO Software Engineering Conference, leading experts declared a Software Crisis, and called for more effective control over wayward software development projects. They wanted better rules for the software development game.

Ever since, plenty of folks have been keen to evangelize *their* rules for software development. Some even package their rules up for reuse, and call them a *methodology*. The promise of a methodology is: "these rules worked well for me, and if you follow them as described they will work well for you too".

Which Methodology is Best?

The problem is, there are many competing methodologies out there. This means there are many competing set of rules for the software development game. The really arrogant have tried to circumvent this competition with a marketing ploy: calling their methodologies "best practices", implying that alternatives are "worst practices" that set your up for failure. Nevertheless, no methodology has yet been established as *the* set of rules for software development.

> As a quick example of rules packaged as a methodology, let's glance in the book Unified Software Development Process (USDP), by Booch, Jacobson, and Rumbaugh, and published in 1998 by Addison Wesley. Here are some randomly chosen rules from that book:

Page 19: “When allocating resources, the project manager should minimize the handoff of artifacts from one resource to another in a way that makes the flow of the process as seamless as possible.”

Page 34: “Developers begin by capturing customer requirements as use cases in the use case model. Then they analyze and design the system to meet the use cases, creating first an analysis model, then a design and deployment model.”

Page 76: “The architecture is created up-front by an architect (during the elaboration phase). This takes considerable up-front calendar time.”

There are hundreds of such rules throughout the book, and the book itself is only a subset of the fuller Rational Unified Process (RUP), which is available for a subscription fee. RUP is claimed, in the preface to the USDP book, to be the very best methodology in the industry.

Everybody hopes they have backed the winning horse, and at conferences and in business meetings, you often hear people get into heated debates about which methodology trumps all others: Is RUP better than SSADM? Is Scrum better than XP? Is Crystal better than Lean?

I am a little embarrassed to admit it now, but I used to fall into the same trap myself. A client would tell me of projects gone astray; I would dig around a bit, and find they were using, say, the Booch method. I would think to myself “Oh, the fools! How could they hope to succeed, playing the game with those outdated rules?” I would then feel mighty proud as I advised them to switch to eXtreme Programming or some other exciting flavour.

Many clients took this advice surprisingly well; at least at first. They would try out some of the recommendations, and in rare cases, even all of them. Some would even make a success of it. As soon as my back was turned, though, they nearly always slipped back into their old bad habits. Not surprisingly, this bothered me. A lot!

Wriggly Advice

At first I blamed the clients for lacking the discipline to follow through. Then I decided that the advice only worked when I was there: I must be some kind of catalyst; a essential ingredient to effective results. This certainly boosted my ego. Later, though, self doubt began to creep in. I wondered whether the problem lay in me giving bad advice. Or maybe I just didn't know how to teach clients to be self-sufficient.

This emotional roller-coaster wore me down, and it made me pretty wary of recommending specific methodologies to companies. So I stopped doing it. I found a sneaky tactic to wriggle away from such discomfort: “There is no best methodology; you have to choose the one that is right for the specific needs of your project”.

Five Issues

Still, I was never entirely comfortable with that wriggly response either, and here are five reasons why:

- People want a single methodology they can use all the time. Expecting folks to be experts in a whole bunch of methodologies, and then able to select the correct one on a project-by-project basis simply isn't realistic.

> I saw this problem first hand at a company in London, where projects were required to demonstrate they had chosen the best methodology according to project needs. In practice, the project team usually didn't and couldn't know enough about a project's specific needs until part-way into the project, so they couldn't sensibly choose the right methodology as the first step. Instead, I saw many teams working under the radar - doing projects any old way – then tidying up near the end with a flurry of after-the-fact paperwork, to fake compliance with whichever methodology would have been most suitable if they had magically known all the facts up-front.

- Different people on a particular project may well have different opinions about which of the available methodologies is the right one for that project. If analysts insist on one methodology, developers insist on using another, and customers want to use a third, the project will likely end up a casualty of methodology wars.

> In the mid 1990s, I consulted to a European bank. For more than a year the bank had relied on external contractors to deliver a new core-banking system. However, several of the contractors constantly demanded ever more detail in the specification of requirements before they would start programming. The remaining contractors were keen to get started with programming, so they could release something soon, and then improve on it when they got feedback. Each side in this battle blamed the incompetence of the other side for lack of progress. The real problem, though, was that the two groups were deeply committed to different methodologies. Each was playing the software development game by a different set of rules. Each was marking the score card differently. What one group saw as

making progress, the other saw as a sure-fire way to slip up.

- No methodology will ever be “the best” for long; better ones will almost certainly come along. Telling somebody that a specific one is “the best” often stops them being open to alternatives in the future.

A large American company - let's call them InterSoft - bought-in a documented methodology from a company which we will call MethodWare. MethodWare had been certified as achieving Level 3 in the Software Engineering Institute (SEI) Capability Maturity Model (CMM).

InterSoft was impressed by this Level 3 certification; it gave MethodWare's methodology legitimacy. It made me twitch though: I explained that CMM was intended to show the process maturity of the organization (i.e. MethodWare), not of the methodology they sold.

The SEI CMM actually defines five maturity levels. Level 1 is essentially uncontrolled chaos. Level 2 is where some useful controls are in place, and are repeated across several projects. At Level 3 the methodology is documented in detail and enforced throughout the organization.

InterSoft saw Level 3 as a final destination, not a milestone in a longer journey. They had been convinced they were now using “best practices” and that shifting from them would be dangerous. Once a company has reached CMM Level 3, though, it is encouraged to strive to Level 4, wherein detailed measurements are taken, and controls put in place, to monitor the effectiveness of the methodology and the resulting software products. InterSoft had no such feedback.

The highest level of maturity is CMM Level 5, where feedback from Level 4 is used to guide ongoing improvement of the methodology, by continually trying out new ideas and measuring their effectiveness. This fell on deaf ears at InterSoft. With their conviction that the bought-in Level 3 process embodied all best-practices, InterSoft remained stuck at a level of capability immaturity from which they would not escape.

- A methodology can never answer every single question. It can never tell you every single rule. Software development is simply more complicated than Monopoly, and Chess, and Football. For each methodology, then, there are many competing pundits, each claiming that their specific and highly detailed embellishment is the one true path to success. Wars erupt about which bits of a given methodology are the most important and must remain intact, which can

and should be tweaked, and which bits can be safely dropped. These conversations quickly descend into heated debated on detailed matters than are mostly irrelevant to project outcome.

I met a team leader at a conference. His team was using the Rational Unified Process (RUP) and its Unified Modeling Language (UML). They used UML collaboration diagrams as part of their design work, but had recently heard that UML sequence diagrams were better. He asked me if they should switch.

Rather than get into detailed discussions about the minor differences between the two types of diagram, I asked if RUP was actually working for them. He confessed that creating all the diagrams actually slowed projects down, and so his best programmers tended to use a tool that generated the diagrams automatically from the software once it was already written.

Digging into this, we realized that the main benefit his team was getting from RUP is that they had learned to think in an object-oriented way, and he was confident this had improved the design of their software. All the other demands of RUP were secondary to that, and his worries about switching from one diagram type to another would probably make little difference to project outcomes.

- The biggest problem by far is that project-specifics don't determine how well anybody will, or even can, take on board any particular set or game rules; people's emotions do. Even if you convince somebody logically that a given methodology is the right choice for their current project, they will resist that advice if it conflicts with their gut instincts about which rules really apply. If you can't get a person to change their emotional reaction, no amount of debating will be of any use. The next chapter has a little quiz that should reveal some of your own gut reactions, and show why some methodologies likely ring true to you, and others sound more like mumbo-jumbo.

Over lunch, a director of software development told me that one of his teams was using prototyping, and had experienced great success with it over the past few months. Despite that, he was unwilling to allow his other teams to do the same, and was even pulling that single team off prototyping and back to a more “traditional” methodology.

I asked why, and he said “I don't buy into these new ideas – they don't scale up to enterprise level issues.” I asked him what these “enterprise level issues” were – and he replied “big companies with lots of teams and lots of projects – in that environment you need the discipline that comes from proper project management – not a cowboy approach

that works for two guys in a garage".

This director didn't actually know much about any of these alternative approaches, but he knew for sure they were wrong for his company – they simply "didn't scale". No amount of convincing was going to change his mind, since it wasn't his mind that needed changing, it was his gut instinct.

Chapter 2 : A Pop Quiz

Gut Reactions to Methodologies

As noted at the end of the last chapter, people tend to have gut reactions to software development methodologies. Maybe you have had this experience yourself: You hear about a particular methodology and think “Is this guy nuts? This will never work in practice!” or you find yourself thinking “At last, somebody is saying what I instinctively knew all along!” Maybe you even had the following very common experience: you came across a book on some methodology and cringe as you read it, only to reread the same book a year or two later and feel like it has all the right answers to project success. It is these emotional reactions to methodologies that interest me.

Pop Quiz

Let's try a pop-quiz. Answer these ten multiple-choice questions, relying mainly on your gut response. Sometimes more than one answer will seem correct, maybe sometimes none of them will. Don't try to be too analytical; there is plenty of time for that later on in the book. For now, just go with what instinctively feels right.

Q1: Which people make the biggest difference to the success of a software development project?

A: Managers: Effective project management can make the difference between those projects that go off track, over budget, deliver late, and produce poor quality software, and those projects that are on time, within budget, satisfy their requirements, and deliver a sound return on investment

B: The Project Team: Management is essentially administrative, and contributes little to the actual delivery of great software, which requires above all a talented and experienced software development team

C: Customers: Without customers there would be no software, since only customers knows what they really need and value, so customers should remain in the driving seat to ensure we continually focus on delivering software that matters to them

Q2: How can we ensure we know the software requirements well?

A: Capture them in detail up-front: Ideally, customers will give us a detailed list of their requirements, or if not we can work with them to nail down their needs early on, providing a fixed and clear target for the project to then deliver on

B: Let them Stabilize: Requirements are at first only a fuzzy approximation of what is needed, but in time clearer details do emerge, enabling us to discover the important and stable requirements, forming a increasingly stable base upon which to build the rest of the project

C: Expect them to Change: Previously important requirements lose their importance over time, and unexpected events lead to the emergence of completely new and unanticipated customer needs, so we should expect requirements to continually evolve, often in directions, and we should remain focused only on those which the customer currently values most highly

Q3: Where are requirements, designs, and other decisions best captured?

A: Textual Documents: Write documents to capture all details of requirements, design, and other decisions, so that information is pinned down, is easily shared with others, and will not be lost

B: Formal Diagrams: Create diagrams, using a relatively formal graphical notation, to express requirements and designs more precisely and concisely than is possible with ambiguous text

C: Human Brains: Collaborate closely with colleagues and customers, to ensure that everybody's evolving mental models of requirements and design are closely aligned

Q4: Assuming we have requirements, what is the next goal?

A: Transformation: Requirements should be translated into a detailed design blueprint, which can then be implemented directly into software code.

B: Model Building: Software always supports some part of the real world, so we should build models that capture the essence of that part of the real world, expressing the most important things in it, and how they relate to one another, and reflect those abstractions in the design and implementation of a robust software architecture

C: Delivering Software: Documents, models, and other interim products are only useful if they help get software into customer hands as soon as possible, so focus on delivering working software quickly and often, no matter how basic at first, then rely on the resulting customer feedback to tell you what to do next

Q5: What can a manager do to help a project progress quickly?

A: Plan: Create a plan, including a time-efficient schedule, then monitor and control the project to ensure it remains on track and follows that plan closely, with all people working hard to live up to their commitments and deliver on their responsibilities

B: Delegate: Put the right people on the project, give them a clear direction, and then get out of the way while they deliver results quickly, without your potentially stifling interference

C: Support: Encourage the team to reflect on their own work to determine what needs changing to make them effective, then continue to provide whatever they ask for, and remove any obstacles they find in their path that hinder their progress or threaten their work pace

Q6: How can we best control project costs?

A: Project Management: Use proven proven management and cost accounting techniques to ensure that all resources are used as efficiently as possible

B: Hire Great People: Hire the very best people, since they are around ten times more productive than average, but cost only about twice as much

C: Focus on Value: Only develop software for the requirements that the customer values most highly, so that you are continually delivering the highest profit possible (value minus costs) at any given time

Q7: How can we ensure that software is high quality?

A: Process: Identify a reliable, trusted, and proven software development process, with a series of fully defined and repeatable steps, each with quality built right in, give each person a clearly defined role with specific responsibilities, then hold people accountable for following that process to the letter

B: Tools: Rather than hampering people with old equipment and outdated principles and practices, provide them with best-of-breed software development tools, and training in the very latest techniques, so they can always follow best-practices and get best-possible results

C: Feedback: Since the customer decides whether the software is good or not, work with them to create measures that detect early if the software is meeting their expectations, and rely on customer feedback to determine what would make the software better from one version to the next

Q8: What is the best structure for an effective software development team?

A: Defined Roles: Each person is assigned to a specific role, such as analyst, programmer, and tester, each with unambiguous tasks and responsibilities, and each contributing a valuable slice to the overall sequence of work required to complete a project

B: Expertise: Leave the architecture to the smartest and most talented technical people, with other experts contributing to whichever area or areas they are best suited

C: Self-Organizing: Motivated people, gelled around shared values and common goals, are encouraged to organize themselves in whatever way they find helps them to be as effective as possible

Q9: What is the best way to prevent unnecessary work?

A: Eliminate Sloppiness: Half-baked wishy-washy information confuses people and leads to backtracking and rework, so ensure that each person lives up to their responsibilities in full, and completes their work to a high standard before passing it on to others

B: Reduce Bureaucracy: Don't waste time on a long chain of documents that few people will ever read, and status reports that are soon stale, rather focus primarily on modelling the core of the business domain to provide continuity and traceability from software code, through design, and all the way back to requirements

C: Don't Do It: Don't work on anything unless the customer sees it as important, since low-priority requirements are likely to change over time and hence not worth up-front investment, and if and when their business value increases they would likely have to be redone anyway

Q10: What indicates that a software project finished?

A: Hand-Over: When it meets all of its agreed requirements and all deliverables have been handed over to the customer

B: Stability: When it is sufficiently stable and trustworthy to be released into customer hands

C: Customer Value: When the customer decides that the next version won't deliver enough extra business value to warrant the cost

How did you Score?

Some of the answers above may resonate, and other may well seem either ludicrous, or mumbo-jumbo. Maybe your preferred answers were scattered all over the place, or maybe they followed a pattern. Most people tend to find they choose mostly As, mostly Bs, or mostly Cs. If that is true for you, then the following may well ring true to you:

Mostly As

- You are likely to be using or at least drawn to a Waterfall methodology, such as the Structured Systems Analysis and Design Method (SSADM)
- Most of the B answers probably seem to over-pamper the whims of the technical folks on the project team, and miss the management oversight necessary to control project risks
- You probably feel that most of the C answers are too touchy-feely, and lacking enough best practices, to have much practical use

Mostly Bs

- You are likely to be either using or at least drawn to a strongly iterative and incremental methodology; probably an object-oriented one
- You may well have chosen mostly As in the past, but have now moved on from that line of thinking, having seen that it sounds reasonable in theory but actually stifles projects in practice
- The C answers probably sound somewhat familiar to you, but are a little too fluffy for your taste, and lack the technical rigour of your own approach

Mostly Cs

- You are likely to be using or are at least drawn to an Agile methodology such as eXtreme Programming (XP)
- You probably see most of the A answers as way out of line with the true nature of software development, and a hangover from the days when people underestimated just how tricky it is to get software development right
- You may well have chosen plenty of Bs yourself in the past, and can still see some merit in them even today, but you now believe they focus too much on technical issues and not enough on the more critical issue of keep up with the customer's ever-changing priorities

Chapter 3 : Beliefs, Values, and Metaphors

Converting People

The pop quiz in the last chapter was a bit of fun, but I hope it also had some value.

Few people choose a random scattering of As, Bs, and Cs. Most people seem to have some gut instinct that guide them to one set of answers over another. Those instincts are important, because they shape what we feel about and how we are likely to react to all kinds of other software development issues. They also help us recognize like-minded people.

If you are a mostly-As person, and you find another mostly-As person, you are going to get on great. But neither of you will have a lot in common with a mostly-Bs or mostly-Cs person when it comes to how to run a software project.

Human nature, though, makes us try to convert people. If a mostly-Bs or mostly-Cs person spends a reasonable amount of time with a mostly-As person, they will find it hard to resist trying to win that person over to their own way of thinking. I am guilty of this myself. I shudder to think how many times I have tried to save somebody from their own blinkered way of thinking about software development.

It is probably the same impulse that compels a new religious convert to want to share the good word with neighbours and friends: most of whom then hope he won't come knocking at the door on a Friday night.

Unfortunately, like that religious convert, you are not likely to have much success in getting others to see the light. More often than not, you will leave shaking your head that they just doesn't get it.

> Not long ago, a software project manager, Lenka, asked me for some advice. She was looking for a tool to help her “Manage the allocation of resources and monitor the progress and completion of tasks”. By “resources” she meant people, and by “tasks”, of course, she meant work.
>
> I cringed. Lenka was using standard project management techniques to manage software development projects. To be honest, I had used the same project management techniques myself for years. Yet here I found myself cringing instinctively at the mere sound of the words “allocation”, “resources”, and “tasks”.

It isn't that I *thought* standard project management was the wrong approach, I *knew* it, deep in my bones. This wasn't my opinion, it was a blindingly obvious fact, and anybody who couldn't that had their head in the sand.

Rather than give in to the urge to rant at Lenka about how foolish she was for taking such a wrong turn, and then ram some vastly superior approach down her throat, I caught myself, bit my tongue, and asked: "So, tell me what exactly you hope to get from it."

Herein lies the problem. If Lenka took the pop-quiz from the previous chapter, she would almost certainly choose lots of As. These days I choose mostly Cs. With my mostly-Cs mouth, and Lenka's mostly-As ears, I would be giving her an unwanted lecture, rather than help she can really use.

Emotional Choices

If you try to convince somebody that one methodology is better than another, they might be impressed by the power and logic of your arguments, and might even go along with what you tell them, but unless they really feel a strong emotional connection to the spirit of your arguments, they will likely be left with lingering doubts; nagging feelings that something just isn't right.

A university professor visited a businessman that I know well. The professor was trying to get funding to research and develop software tools that enabled software specifications to be entered into a computer using a very precise notation. The idea was that the tools would then execute these specifications directly, with programmers having to write only very small amounts of code to fill in minor details that could not be gleaned from the specification alone.

The businessman called me afterwards: "That professor is a very smart guy. His arguments were very persuasive. But, the whole thing doesn't sit well with me. I just don't buy it." He turned the funding request down based on his gut instincts.

Why is it that some people make the leap to a different methodology immediately, some can take years to make the same move, and others refuse to budge no matter what?

Over the past twenty years or so, I have come to believe very strongly that methodology choice is rarely a logical decision, but more commonly an essentially emotional one, based on deeply held beliefs wrapped up in values and metaphors. These values and metaphors dictate what we feel software

development is really all about, and so influence our emotional response to every software development methodology we come across.

Beliefs

Different people have different beliefs. That is pretty obvious, but bear with me while we dig into it a bit. A belief is something you *think* is right. If you come across something that confirms your beliefs, you are going to agree with it intellectually, and nod your head. If you come across something that conflicts with your beliefs, you are going to disagree with it intellectually, and shake your head. You might be tempted to argue back. You might even be able to convince somebody that you are right and they are wrong, changing some of their beliefs.

Beliefs start out as acts of faith. Somebody tells you that some rule applies in certain circumstance, and you believe them. This is fine for a beginner: they trust that the creator of some methodology knows more than them. Beginners need to be handed concrete rules to live by. Anything overly "philosophical" will be too abstract to put faith in and act on. Many methodology books focus on this audience, and they describe a given methodology in terms of a list all the rules and regulations that must be followed. With a little persuasion, beginners will have faith that those rules and regulations are right, and accept them as facts.

> PUT SOME EXAMPLES
>
> previously this talked about RUP claiming to be the best in the industry. It is certainly one of the most dogmatic, that is for sure, which, again, is excellent for beginners who seek definitive rules.

I have come across dozens of methodology books over the past twenty years or so, full of firm rules that make them look very authoritative: "If you follow these rules religiously, you will succeed". When I first started out in software development I would find these thick rule books far more reassuring that thinner book that offered only general advice.

I have seen beginners question the validity of rules in a methodology book, instead they usually ask for specific points of clarification, such as "How quickly should a customer receive bug fixes?" and "Which documents should testers create for weekly status meetings?". That is, beginners are uncomfortable with uncertain rules, and seek concrete answers to fill in any gaps in the rule book. Knowing their audience, many methodologists are sensitive to such gaps, and they are highlighted,they tend to be filled in the next version of rule book. The results is that the rules become more intricate, and hence the methodology more dogmatic over time. This certainly makes the methodology less subjective, which again is great for beginners, but the danger is that it can also make the methodology

less flexible and eventually too heavyweight to follow, which is bad for everybody.

> In the late 1990s I spent some time with Derek Coleman, one of the authors of the then-popular Fusion methodology. He confessed that the current version was now hundreds of pages longer than the original, with so many intricate rules that he could no longer see the wood for the trees himself, and was therefore swamped by the complexity of his own methodology. Derek was deeply worried that the Fusion methodology was collapsing under its own weight, and decided to take it no further.

Values

Beliefs are fine if the rule book is right, but the danger is that the rule book can be taken so literally that it is followed for its own sake, without asking whether or not it is actually working. That might fine for beginners, but it usually isn't good enough for more experienced people. Rules are based on faith. Faith is belief without evidence, and faith shouldn't be questioned. Intermediates need more than faith, though. They want proof. If some rule sounds right, intermediates may give it the benefit of the doubt and accept it as a to-be-proven-belief, but emerging evidence on whether or not the rule works in practice will either affirm or weaken that faith. Beliefs that continually prove ineffective will eventually weaken to the point of collapse. Beliefs that are reinforced by repeated affirming experiences, will eventually become values.

Whereas a belief is something you *think* is right, a value is something you *know* is right, deep in your gut, because it has proven itself to you time and again. From then on, whenever you come across something that confirms your values, you are going to have warm fuzzy feelings about it, and it will cement your values even further. If you come across something that conflicts with your values, you are going to have an uncomfortable emotional reaction to it. What you heard won't *feel* right, and will make you anxious. These emotional reactions are much stronger than the intellectual ones associated with beliefs. Arguing back isn't going to help much, since it will just make the other person feel anxious too.

Although you can convince somebody to change their beliefs, it is much harder to persuade them to change their values. People can really only change their values themselves, and therein lies the problem. When different people hold conflicting values, it is very hard for them see eye to eye.

> It was my thirtieth birthday. I was working on a short term assignment to a company called Sheffield Insulations, in Sheffield, UK. The manager of the software department there said to me “You do great work, but I would never employ somebody as old as you. Folks with

your level of experience are too stubborn. You can't teach an old dog new tricks. I want people who work my way, not their way."

Sheffield Insulations, and indeed many companies, like to employ beginners. Partly because they are cheap, but more so because they will not have enough experience to have developed values about how software should be developed. They may have beliefs, but these can be easily changed. Beginners are great, because you can impose your own values on them. Your values become their beliefs. They will follow your rules. The downside is that beginners need a lot of hand holding, and can't bring with them experience that you are missing. Still, if your main priority is compliance, go with beginners.

Our values are the things we most care about. We use them to determine if we have been successful or not. We also use them to measure the success of others. Naturally, everybody wants to succeed, but if different people have different values, they will each measure success differently.

When people measure success differently, you get conflict. People's values don't need to be wildly different for conflict to occur. You only need a few values to differ for there to be problems.

As an example, let's imagine that everybody in a company shares the following value:

- Whoever knows best about something should make decisions about that thing

In fact, let's imagine that this gets written up as a "company value" and posted on the wall for all to see. There you go, we now know who will make decisions in this company.

But wait! Now imagine that managers also hold the unstated, private value that:

- The boss always knows best

Whereas many others in the company hold the unstated, private value that:

- Project teams know best

Well, you can imagine the conflicts when it comes to decision time.

Methodologies and Values

So, values are important, since they determine how we feel about things, and they drive our actions and reactions. Some methodology authors have picked up on this, and captured the values underlying their own methodologies. For example, a group of Agile methodologists have agreed that they share the following four values:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

The idea is that those methodologists value the left hand side of each sentence (e.g. “Individuals and interactions”) more than the right hand side (e.g. “processes and tools”).

Just like the company value statements mentioned above, this sounds nice, if a bit abstract. However, just like a company value statement, this is marketing material for Agile methodologies. It doesn't tell us what each of the authors really values. For example, if I asked one of the agile methodologists “Is it acceptable if the manager makes all decisions?” each would likely have a very emotional response, almost certainly followed up with “no – teams must be empowered to make their own decisions”, yet the underlying value that led to this isn't in the list above.

I don't believe methodologists, or most folks for that matter, are being deceitful and “withholding” their real values. More often than not, when I have pressed people on this, they couldn't actually articulate their values even if they tried. Sure, you can sit down and tease out a list of values – as I have done several times with people – but that always seems to be something going on beneath the level of the values.

Heuristics

Beginners, then, rely on beliefs, which give them clear rules to follow. Intermediates have internalized beliefs that proved effective, forming personal values. Experienced folks, at least in my experience, are a bit different. Sure, they still have beliefs and values, but beyond that they rely less on rigid rules, and more on rules of thumb. That is, heuristics. Somebody experienced knows when not to apply a rule. They also know when a rule needs adapting to new circumstances.

> There is a great story in the book Peopleware, by Tom Demarco and Timothy Lister, about a form of strike called work-to-rule. This is where workers follow the officially imposed rules (the procedures manual) to the letter, and of course progress grinds almost to a halt. Knowing when and how to work around the rules is essential to working effectively. This is where experience comes in to play.

Over the years, I have sat down with many experts and tried to tease out their heuristics. It proved surprisingly difficult. All too often, I would hear rambling heuristics like: “Well, a general guideline is

that software developers should do their own testing, but at the same time customers should help them decide what the tests should look like, but in practice that might not work well because customers aren't used to working in this way, so sometimes the developers have to write the tests and ask the customers to review them, but then that depends because often the customer doesn't know what the tests should look like until after they have played with a working system, but that depends too because the tests are really the requirements for the software and you can't create a useful system without them, but that depends because sometimes prototyping is the only way to get any feedback at all, but that depends because management might not allow it, and even if they do, it depends because the customers might get the false impression from the prototype that the software is nearly do. So, overall, it depends."

Well, heuristics like that aren't much use to anybody!

So, experienced folks do a sneaky thing. They boil their heuristics down to a set of simplified rules. They write these rules down and call them a methodology. The bold ones then claim this methodology is *the* set of best practices, then make a nice consultancy career helping confused people deal with the sticky bits. The more faint of heart know they are misleading beginners and intermediates, and add a sneaky disclaimer to their books "Modify according to your specific project needs". This, of course, is a euphemism for "It depends". That's about as helpful as the cooking instructions that came with some very expensive pasta I bought in Rome: "Cook until ready". You still need the author to tell you what "ready" is. It is no surprise that more than once I have heard cynics claim that methodology books are essentially marketing material for their author's consultancy business.

Metaphors

In a sense, then, experts don't follow the rule book, they write it. Sometimes literally. But they don't exactly follow the rules they have written down. Instead, they rely on complex heuristics.

As I have become more experienced myself, I have thought about my own heuristics. I tried to write some of them down. It wasn't easy, and lots of them also looked spookily like a long winded way of saying "It depends".

To be blunt, I don't believe my brain is smart enough to have a long and rambling list of heuristics, and I am sure I am not processing such a list on the fly when making decisions. Well, maybe sometimes I do find myself thinking through some lose rules I have collected and internalized over the years, but more often than I find that something else is going on. What I have discovered is that I have abstracted many the rules and heuristics into a set of deeply held metaphors, which I continually elaborate. These metaphors feed my decision making. These metaphors can be thought of as generators for values and beliefs. Metaphors are much more active than lists of rules, since they are open to continual elaboration and adaptation to changing circumstances. Metaphors, then *are* heuristics.

Once I had come to believe that metaphors underlie my own values and beliefs about software development, I became obsessed with them. I annoyed the heck out of people for weeks, trying to get them equally excited about them. Some folks went along with it, but few had the same excitement as me. One of my friends explained this rather wittily: “You can convince folks to believe in metaphors, but only they can change themselves to value them. You, Anthony, have become obsessed with the metaphor metaphor”. Ouch!

Undeterred, I stopped trying to convince folks logically, and started helping them uncover their own metaphors, and hence convince themselves. What I discovered, unsurprisingly, is that experienced people did have plenty of metaphors they called on. What was surprising to me what that it didn't matter much in which area the person was experienced, they would still be driven by metaphors they had developed over time. For example, I came across folks who had never worked in the software business, and had switched industries, and relied heavily on the metaphors that worked in those prior industries.

I met several managers with MBAs and finance backgrounds, with metaphors strongly based on cost management, and it showed in the way they ran projects. I met a restaurant manager, now a project manager at a software company, whose principle metaphor was still about keeping the chaos of the kitchen (here, software developers) hidden from diners (here, managers and customers).

The more I dug into this, and the more people I worked with, the more evidence I gained that experienced people run software projects according to deeply help metaphors. The ongoing elaboration of those metaphors in changing circumstances deepens their experience, and increases their commitment to those metaphors.

A simple dictionary definition of a metaphor would be “One thing with which you are familiar used to conceive another thing with which you are unfamiliar”. An example would be telling somebody unfamiliar with your boss that “My boss is a clown”. Now, unless you are a junior clown, this statement is a metaphor, meaning that things which are true about a clown are also true about your boss. This helps folks who don't know your boss understand that he makes you laugh, he is clumsy, he keeps making a fool of himself, and so on. Of course, different people can interpret this metaphor differently. For example, somebody might decide that clown is an actor, therefore your boss is only acting foolish, rather than actually being foolish. It is this openness to personal and ongoing interpretation that makes metaphors so useful, as we will soon see.

There is plenty of research into metaphors, and it turns out that we use them all the time. They are so pervasive that we usually don't even realize we are using them. A splendid little book called Metaphors

We Live By, written by Lakoff and Johnson shows how we start out as infants learning basic spatial metaphors, such as up being good, and down being bad. We hold on to these for the rest of our lives: “It's time we started living the high life!”; “His chances of a promotion are going downhill fast!” We learn that ideas have directions: “I see where you are going with this!”; “I just don't get where that guy is coming from!” We then spend the rest of out live creating new and increasingly rich metaphors, layered and connected in intricate ways, and these feed the way we think and feel about the world, and help us explore ideas and share those ideas with others.

Metaphors work because they have implications. Exploring those implications helps us understand one thing in terms of another. When we first heard people talk about “The dot com bubble” we understand exactly what they meant: bubbles inflate rapidly, they burst, and we see they are full of air. We can see the connection. But if somebody had mentioned “The dot com hamburger” we would probably struggle to connect the implications of hamburgers with what happened to dot coms. Hamburgers are a pretty week metaphor for dot com.

If the implications of a metaphor are positive for us, we will likely accept, enjoy, and continue to explore the metaphor. If the implications are uncomfortable, we are almost sure to reject it. That is why many computer programmers smile knowingly when one says to another “My manager is Dilbert's pointy-haired boss” but most senior managers would cringe at the same thought. For metaphors to work, then, their implications have to resonate for you personally.

Once you have found a good metaphor for something, you are unlikely to let go of it, unless it is overthrown in some deeply emotional way. That is, you usually have to be shocked out of it. That means it is extremely hard to convince somebody of something if it conflicts with a metaphor they cling to. For example, saying “Software developers are creative puzzle solvers” won't get a very positive response from somebody who says “Software developers are glorified typists”, and vice versa. The implications of the two metaphors are simply in conflict. No amount of badgering is going to change the other's mind. If anything, you will just make them more convinced that they are right and you are wrong.

Three Metaphors of Software Development

So far, I have identified three very common metaphors of software development:

- Production Line: A project is run like a factory. Requirements are pinned down in an up-front plan. Workers are stationed in sequence along a production line, each following specific tasks and a time-schedule detailed in the plan. Each worker passes their output along the production line to the next worker. The finished product arrives at the end of the production line. Managers are there to ensure that people adhere to the plan, and punish those who do not since they

threaten project success.

- Model Building: Software exists to support humans in some part of the real world. This requires knowledge of that part of the real world and how people work in it. This is best achieved by building scale models, which become the architectural core of the software system. Naturally, the real world changes over time, but well thought-out architecture will have anticipated future changes, and thus be able to absorb them. Workers need considerable expertise to be able to create good software architectures. Managers are there to interact with customers and help expert workers determine customer needs.

- Customer Satisfaction: Software must be valuable in the eyes of its users, and the folks who pay for it. Therefore, they should decide on the order in which software features are developed and delivered. Delivering often allows them to: use features while they still have high value, change priorities for which features are delivered next, and halt further development when costs outstrips potential value returned. Software teams should do whatever it takes to satisfy customers. Managers should teams do this, by supporting them rather than controlling and stifling them.

The Production Line metaphor tends to appeal to people who chose mostly As in the pop quiz, and the Model Building and Customer Satisfaction metaphors tend to appeal to people who chose mostly Bs and mostly Cs respectively.

We will get into the implications of each of these metaphors later. For now, just dwell on the names and the very brief descriptions just given, and ask yourself how they relate to software development. Do any of them resonate, or cause you an emotional reaction, either positive or negative? Maybe they do, maybe they don't. Certainly, though, by the end of this book you will understand, and hopefully even feel, each of these metaphors very well.

Only Three?

A word of caution: These three metaphors made it into the list because they have proven to be the most prevalent and robust. I can't claim these are the only metaphors of software development. There certainly are many others, and section XXX of this book surveys some of them. If and when these other metaphors gain mainstream popularity, later editions of this book will update the list of three to include them.

> A salesman I came across seemed to view projects as liquids. He said of one project “We are going to keep milking this cow for years”, and about another “The company should pull the plug on it.” I wondered if I had come across a new software development metaphor,

> until it dawned on me that he wasn't talking about the projects themselves, but about money earned or spent. He was talking about liquid assets.

Negatively Charged Metaphors

Metaphors, are highly persuasive. We have to be careful how we choose them, and use them rather than abuse them. Some metaphors are inherently emotionally charged, and if misused can turn us against a particular methodology, or even a whole school of thought. For example, describing a particular school of thought as a Black Hole is only going to inspire negative thoughts.

There are plenty of metaphors like this that people use to capture and reinforce negative feelings, often for methodologies or schools of thought they don't known much about, and these metaphors can be pretty infectious and highly persuasive to others. We need to guard against them.

> I have heard people disparage the Production Line metaphor by calling it the 1984 approach to software development – bringing to people's minds the horrors of George Orwell's book of the same name. I have known people who become fearful of the Model Building metaphor as others called it the Mad Scientists Laboratory – bringing to mind images of out of control lunatics pursuing their own dangerous whims. Most recently, I have witnessed people scared away from the Customer Satisfaction metaphor by hearing others call it the Wild West Road to Chaos.

Positively Charged Metaphors

Of course, the same trick can be used to bring an inherently positive bias.

> I heard a salesman calling the Model Building metaphor Latest and Greatest – which doesn't tell you much about how it approaches software development, but may well give a customer warm and fuzzy feelings. Likewise, one of my managers always called the Production Line approach Proven Best Practices. You can guess how a senior executive reacted when asked to choose between Proven Best Practices and the Wild West Road to Chaos.

Neutral Metaphors

When we hear people using a metaphor we should ask ourselves whether or not it is inherently emotionally charged. If so, it is likely to color their own judgement, and the judgements of others, in a heavily biased way.

While preparing for this book, as people described to me their own views of software development, I learned to pay close attention to emotionally charged language, and the metaphors that feed it, and tried hard to steer the conversation back to more neutral language. Hopefully, I have been reasonably successful with this, and the three metaphors presented here are not in themselves either positively or negatively charged. Naturally, people's individual emotional reactions to one or more of the metaphors may be overwhelmingly positive or negative, but my hope is that these reactions stem from their personal interpretations, rather than an emotional bias in the metaphors themselves.

Mixed Metaphors

Each of the three metaphors, colours our judgement and guides, or blinkers, our lines of thinking in deeply emotional ways. Anything that matches the metaphor we subscribe to will be readily accepted. Anything that contradicts it will be generally rejected as not ringing true. When you have a discussion about software development with somebody whose core metaphor differs from your own, almost everything each of you says to the other will conflict. Most likely, you will each feel that the other is a crank or a has-been.

Somebody who shares the same metaphor as their colleagues will be seen as a “team player”, but in group dominated by a different metaphor they would be seen as a maverick. If a project has people working under the assumption of different metaphors, that project is suffering from mixed-metaphors.

In linguistics, a mixed metaphor mixes two unrelated metaphors together. For example: “The project manager kept our ship on steady ground”. We can work out what is probably meant here: The project manager is seen as a ship's captain. Presumably, the ship is the project, and steady ground means that the project progressed smoothly. Yet ships on steady ground are not going to get very far. Ships only make progress on water.

Mixed metaphors might seem relatively harmless, but all sorts of assumptions are going on underneath. If some people on a project team think of projects as sailing into uncharted waters, and others see projects as making steady progress along solid ground, they will have very different expectations about how people should behave, and what counts as success and what counts as failure. For example, the sailing folks will likely expect to face an occasional major crisis on a project, seeing crises as rough seas and calling for all hands on deck to deal with them. Success, then, is dealing effectively with the inevitable crises that projects are bound to face. Failure to these folks comes from naively expecting

smooth water the whole time, since rough seas are inevitable, and panicking when faced with them is the surest way to sink the ship. The steady progress folks, on the other hand, will likely feel very anxious about a crisis, seeing it as a major failure to plan ahead for a safe and steady course.

NEED TO PUT AN EXAMPLE HERE

Mixed-metaphors, then, are not just an intellectual curiosity. They can cause very real problems to real projects. People on the project will have a hard time accepting what each other is saying, and an even harder time working together effectively. Quite simply, they will have very different expectations about how to run the project, and what it means for the project to succeed.

In chapter XXX we will look at a popular mixed-metaphor methodology, which trys to keep everybody happy by squishing together all three of the software development metaphors. We will see that it has bits of Production Line, bits of Model Building, and bits of Customer Satisfaction all mixed together. Does this seem like a good idea? You can take at peek at chapter XXX to find out.

More Than Just a Name

A metaphor is much more than just a name. If this were an academic paper, I would probably say something like “a metaphor is a context-sensitive theory who constitutive implications guide a personal and on-going process of elaboration of a constructed mental image and the perpetual construction of families of candidate elaborations”. Let's break that down into pieces:

Context-Sensitive Theory

The insights from a metaphor don't come from its name, nor from a written description of it, but from the mental activity that it triggers. A metaphor, then, is a theory. Not in the common meaning, where a theory is just an opinion. But in the mathematical sense where it defines a set of consistent and truthful statements that lead from one another. That's all a bit abstract, but what it boils down to is that as metaphor isn't shorthand, but rather a vehicle for elaboration and discovery. The name of a metaphor just gives you the starting point. From that starting point, the metaphor weaves together a whole bunch of concepts and relationships, and guides you through the process of investigating and elaborating them in whatever context you choose. For example, if we accept that software development is a production line, then a whole bunch of things that are true about production lines must also be true about software development. Since production lines pass work along a sequence of task stations, staffed by workers, then software development must have those too, and so on.

My brother is a software developer, and was asked by an elderly relative what his work involves. He replied: “In essence, I solve crossword puzzles all day”. I am not sure about our elderly relative, but I immediately grasped exactly what he meant: All the parts of the overall solution should fit together neatly; sometimes clues are cryptic; sometimes they are easy riddles; there is the frustration of the last elusive clues; the thrill of a slippery answer finally coming to you; and the satisfaction of the whole puzzle finally solved.

Constitutive Implications

The basic principle of metaphors is that concepts commonly associated with one thing, are carried over to another. These carried-over concepts are called the implications of the metaphor. They allow us to approach, explore, reflect upon, and discuss a new area in terms of one we already understand. Saying that a methodology is the Rational Unified Process is far less evocative than saying it is a Model Building methodology. The metaphor forces our mind to reach out for the implications of model building, and transfer them over to software development. Model building gets us thinking of scale models, simulating something larger and more complicated, to draw out the essential aspects of that larger thing.

A colleague had heard of a new book on Domain Driven Design. He thought it might be interesting, and asked me what it was about. I explained it meant learning a business area well enough to build a scale model of it in software. I didn't have chance to explain any more than that, since he grasped hold of the Model Building metaphor, and by following its implications, was able to give me a long lecture on his newly discovered and surprisingly deep knowledge of Domain Driven Design!

Personal Process of Elaboration

A good metaphor touches us deeply, and is open to interpretation at varying levels of complexity, starting from an initial impression, or vague feelings, forming a skeleton – a conceptual scaffolding - which is then gradually fleshed out and elaborated over time. This process of elaboration invariably leads to new internal metaphors, with new vocabulary and new relationships, which themselves are subject to a personal and ongoing journey of recursive elaboration. These discoveries are the “Aha!” moments, as the metaphor guides you to fresh and exciting insights. In short, good metaphors are great teachers: they make you think.

Standing at a whiteboard, I helped a project manager tease out and draw a mind-map of his own internal metaphors of software project management. Pretty early on he was pondering project risk control, and said “the biggest risk is that the customer doesn't want it”. This lead his to ponder customer values, and rethink the backlog of customers requirements. Rather than viewing it as a long list, he started to think of a bottle of milk with cream rising to the top. He talked of skimming the cream of the top. New metaphors were clearly unfolding in his mind, and it was delightful to see his excitement grow as he unravelled them personally.

Constructed Mental Image

A metaphor, even when elaborated, isn't likely to be remembered as a long string of words. Metaphors aren't at their most useful when they are written down, or repeated word for word, but when they reside in your mind, absorbed as rich mental models from which further insights can spring. The most effective metaphors are likely quite sparse in terms of words, but they have strong imagery value. Writing a fifty line description about software project management, including the observation that “it involves coordinating the activities of software development staff towards completion of items from a product requirements backlog which have been prioritized in terms of the order in which customers rank their importance”, is more complete, but far less evocative, than saying that software project management is about customer satisfaction.

Psychologists have demonstrated repeatedly that low imagery metaphors can only be retrieved by the mind slowly and serially, rather like memorization and parrot-fashion repetition of the alphabet. Since the alphabet has low imagery value it is difficult to remember initially, and then is only easily repeated in the sequence in which was learned. Repeating the alphabet backwards is much harder. High imagery metaphors, on the other hand, are remembered more rapidly and for longer, are comprehended more fully, are recalled more quickly and in more highly flexible ways, and support a much greater number of mental cues which can be explored speedily and thoroughly.

This does not mean that a metaphor should be expressed primarily as a drawing or other image in order to explain it to somebody. Metaphor experts have found that drawings do not help communicate and teach metaphors to others. It turns out that far more important than accompanying a metaphor with a drawing is that the metaphor itself is high in image invoking value. An effective metaphor needs to be described in a way that invokes in the audience the construction of personal vivid, memorable, mental image, constructed against the backdrop of the audiences individual background, knowledge, and experience – an image high in personal allusive ties that stimulate the audience's own imagination.

When I first heard about methodologies for which I use the metaphor Customer Satisfaction, they were generally known as Lightweight methodologies. The originators of this term hoped people would understand it meant low on bureaucratic ceremony. However, many people associated Lightweight with images of flimsiness, and from this gathered that the methodologies were without substance. The methodologies were quickly relabelled Agile. Unfortunately, I have noticed that Agile leads many people to immediate images of projects finishing very quickly. This gets them overly fixated on speed, and the hunger to go ever faster. Although these methodologies can indeed help projects to go quite quickly, that is not their central focus. This is why I prefer to use the Customer Satisfaction metaphor, since I have found it more often results in newcomers gaining mental images better aligned with those of people who already use Agile methodologies.

Families of Candidate Elaborations

A metaphor's primary value is in its ability to inspire us to elaborate it. When we stop elaborating a metaphor, it is essentially dead. An active metaphor is always open to further elaboration. Sometimes, a particular frozen elaborations can become widely accepted as an established idioms. Individual software development methodologies are idiomatic elaborations of the metaphors of software development, yet even then many people just go with the spirit of the methodology, elaborating them further to suit their own needs.

When a metaphor is brand new to us, it has no elaborations at all. At first, we tend to start with a single, highly literal, elaboration: a production line has a fixed time schedules for each task, therefore software development must have the same.

A company that publishes statistics hired me for two months to help a senior software architect with some particularly tricky design work. He was using an graphical tool to draw diagrams of his designs. Before looking in detail at those designs, I turned the talk to requirements. As I learned about requirements, I opened an editor, and started typing in some code: a small test, capturing a small requirement. The architects warned me that if the head of the department found me typing in code, I would be reprimanded severely. Architects weren't allowed to write code; they drew high level designs, which were then passed on to senior developers, who created detailed designs, which were then coded-up by junior programmers. This literal interpretation of the production line seemed pretty crazy to me, since the architects had no way of knowing that their designs were any good, apart from gut feelings. I brought this up with the head of the department, and he cut me off,

announcing that the company's approved methodology was unambiguous: architects do not write code.

As we gain more experience with a metaphor, we start to form a whole family of elaborations, with each family member inspired by and suited to the specific circumstances of new projects we come across. Each decision point in an elaboration can result in a new branch of the family tree, and each new insight can ripple up and down the family tree, further elaborating and enriching established family members.

It is this experience-led population of a family of elaborations that transforms a methodology novice into a methodology expert: not just following a single elaboration slavishly, but have a rich body of elaborations to draw on and further elaborate.

Selima Software worked with my company to develop a new version of their software system using a Customer Service methodology. This requires constant access to customers, to determine requirements and their shifting priorities.

No customer was available full time. We started with part time access, but soon found ourselves waiting around, while customers tried to find time to meet with us. Sean, one of Selima's own employees, agreed to be a proxy customer. Wherever possible, we would speak with real customers, but on a daily basis, Sean would substitute for them, and it was his responsibility to ensure he represented them well.

Sean turned out to be an extraordinarily valuable person. With a deepening awareness of the customer perspective, and increasingly detailed knowledge of the company's software products, he became an ideal bridge, appreciated by both groups. So good, in fact, that even when a project has regular access to real customers, I now ensure we cultivate people like Sean too.

At the most extreme, people cling to an unsuitable metaphor, and warp it almost beyond recognition, to meet the needs of projects for which it is a poor fit. At this point, we should almost certainly switch to a whole new metaphor.

A friend and colleague, Bob, was writing a compiler for the programming language BASIC. He was told to copy a particular flavor of the language, for which a programming manual was available to determine syntax, and a competitor's implementation was available

to determine behaviour.

Bob believed in the Model Building metaphor, and as he learned more about the language flavor, he built an increasingly stable software architecture to to suit its implementation. As he neared completion, a few important customers were brought in to have a look-see. One mentioned to a salesman that he liked Bob's work, but it wasn't the flavor of BASIC they wanted. Bob was sent a programming manual for this new flavor and told to make the minor tweaks necessary to support that flavor too. Of course, the tweaks weren't minor; they completely destabilized Bob's architecture.

Having announced earlier that his architecture was nearly complete, management now had a hard time accepting that Bob needed to rework everything. Bob, on the other hand, fumed that management had moved the goalposts, undermining all his progress so far.

Bob worked night and day to re-architect his compiler, to now suit the new flavor too. Eventually, he delivered a version that supported both flavors well. His architecture was now finally stable. Until, that is, a new customer came along with demands for a very different variant of BASIC, that completely upset Bob's hard-won stable architecture all over again.

Everybody had started out with good intentions. Requirements were expected to be pretty stable, if not fully understood at the beginning. The Model Building metaphor seemed like a good match – at least until reality hit. The company was driven by changing customer needs: potential sales overthrew previously stable requirements, shifting priorities in unexpected ways. In circumstances like that, stable architectures aren't going to remain stable for long. Bob, and the company as a whole, would have been better off shifting their metaphor from Model Building to Customer Satisfaction.

What's the Practical Value?

Now might be a good time to revisit a question we asked in the preface of this book: This metaphor stuff might be interesting intellectually, but is there any real practical value in it? Here, we will elaborate on the answers given in that preface.

When you unravel the three software development metaphors you can:

- Better understand yourself, and why you feel comfortable working in certain ways and resist

others instinctively

- See where other people are coming from, leading to mutual understanding of each other's perspectives, rather than dismissing one another as fools who just don't get it
- Realize why logic and intellectual arguments won't convert anybody to your own way of thinking, if what you are saying causes negative gut feelings in your audience
- Gain deep insights into the various schools of thoughts, rather than the superfluous distractions of individual methodologies
- Have a foundation from which to evaluate and understand new methodologies as you come across them
- Gain a wider range of perspectives with which to view a project, realizing that your own single perspective may be limiting your judgments
- Help a project team explore their own metaphors, gain insights themselves, and see new ways of working to which they may previously have been blinkered
- Act as a mediator, bridging differing perspectives, and seeking commonality and mutual respect, where previously turf battles had hindered project progress

Chapter 4 : Managing Teams

This chapter may be moved to another section

Introduction

Scott Adams has been very successful with his Dilbert cartoons, showing incompetent managers frustrating their teams. Alas, this comedy does seem to reflect some reality, at least in the software industry. We have all heard folks in software teams complain that “we need less management and more leadership around here”.

What is going on? In my experience, folks aren't saying that they don't want managers, but they do want a less hostile relationship with their managers. That isn't because they are spoiled children who want to have fun without responsibility. It is because, quite frankly, a lot of tough-guy management is detrimental to projects. A hostile manager might boost his or her own ego and status, but more than likely will thwart team success, so it is no surprise teams don't want them. What teams do want is supportive management; managers that help them succeed on project, rather than sucking them into time-and-energy sapping power struggles.

Imposing Metaphors

In the last chapter we talked about beliefs, values, and metaphors. We saw that you can convince somebody to change their beliefs, but their values and metaphors are more deeply ingrained emotionally and therefore harder to change. All of this matters because it affects the relationships people form, the work they do, and how they measure their own and each others success Methodologies affect precisely the same things. Therefore, your particular values and metaphors will draw you instinctively toward some methodologies and repel you emotionally from others.

We saw that things go well when all members of a project team are committed to the same metaphor, whereas mixed-metaphor teams are likely to experience internal conflict and this affects project outcomes. Even when all members of the team are gelled around a common metaphor, they can still be stifled by the metaphor to which their manager is committed, and vice versa.

For example, a manager who holds on to the production line metaphor, will likely value command-and-control, expecting to make all decisions and give orders to workers. Success will be measured in terms of how perfectly workers obey these orders. If that manager is in charge of a team who shares the same metaphor, things will go smoothly.

If, however, the manager is faced with a team that has gelled around a different metaphor, that team likely resist the manager's orders and want to do whatever they think is right to make the project success. In effect, they will try to bypass the manager they see as incompetent. The manager, on the other hand, will likely see the team as endangering the smooth flow of the production line and leading the project to eventual chaos.

In short, if everybody in a team is committed to the same metaphor, then things will run relatively smoothly. Put in a manager who resists that metaphor, and you have put a cat among the pigeons.

> One of my clients had a team that had worked well together for years. Senior managers, though, were worried, though, that the team might be stuck in old habits, and could benefit from some new blood. They decided to bring in Jacques, whose resume was certainly extremely impressive. Jacques seemed to have plenty of great ideas that promised to shake things up and get the team working in new ways.
>
> Jacques shaking up, though, turned out to be more like an earthquake. Jacques was a strong adherent of the Production Line metaphor. His command-and-control mentality was entirely disruptive to the team, which had thus far relied on the collaborative approach underlying the Customer Satisfaction metaphor.
>
> Jacques couldn't understand why they were so undisciplined, and was deeply frustrated that they constantly veered off track. The more they resisted, the harder Jacques tried to control them. The team couldn't understand why Jacques was such a tyrant, and why he kept forcing them to do things that distracted them from working well together.
>
> The clash of metaphors showed in results: projects slowed down, motivation dropped, nobody was happy. Jacques blamed the team, and the team blamed Jacques.
>
> Eventually, the team simply refused to work with Jacques any longer. They demanded that senior managers brought back their previous manager, who was committed to the Customer Satisfaction metaphor. Projects started to improve. Jacques was sent packing.

It is quite common for teams and their managers to hold on to different metaphors. For example, professional managers with no technical background, are often drawn to the Production Line metaphor and administrative management. Software teams with highly experienced technical people are likely to have gelled around either the Model Building or the Customer Satisfaction metaphor. Usually, the outcome is less dramatic than with Jacques. Instead, teams and managers with conflicting metaphors tend to just struggle along, each frustrated with the other, and each dismayed at the mediocre results that inevitably emerge.

A team of mine was hired by an ex-accountant, now a senior manager. They were told to sit with human-resources experts to get a grasp for the human-resources business area, and write a new software system to support it.

After three months, they gave a demonstration of progress, and were asked “When is it actually going to be finished?”. A senior guy in the team replied: “It is up to you to decide when it is good enough to release to customers. In the meantime, we will keep improving it for as long as you want.” This carried on month after month, with the team continually releasing new versions of the software, but the manager unwilling to release anything until it was finally “done”.

About eighteen months into the project, a wide selection of human resources managers were invited to a workshop, at the end of which several of them asked for immediate installation of the software in their own departments. One said “It might not be perfect yet, but it's a lot better than the software we have now.” Still, the manager would not release the unfinished product.

The team grew increasingly frustrated that the manager could not accept that software was never “done” - it just keeps improving over time. The manager grew increasingly impatient with the unwillingness of the team to finish what they had started.

This stalemate carried on for almost three years, until the manager finally bowed in to increasing customer pressure, and shipped the product a select few, despite his misgivings that it was still not finished.

These ongoing conflicts don't come down to laziness or stupidity. They come down to the fact that the team and the manger subscribed to different software development metaphors. The software development team believed in the Customer Satisfaction metaphor: seeing software as something that is never finished; software is always soft, and therefore evolves continually as customer requirements shift over time. The manager, perhaps because of his accountancy background, saw software development as a one-off production run: subscribing to the Production Line metaphor of turning requirements into completed code. He was looking for a tidy end point, where he could draw the line under the requirements and finally close the book on the software development project.

Now, here is the thing. Managers have more authority than the teams they manage and can easily force his or her own metaphor on the team. This leaves the team with two choices. Either rebel, or comply. If the team rebels, the manager can punish them, thus forcing the team to comply anyway.

When a team complies against their will, they may well do as their manager commands, but they will experience a great sense of anxiety. The manager is preventing them from working the way they deeply believe ensures project success. The have switched their focus from making the project succeed, to simply pleasing the manager. Success is now measured in terms of compliance, rather than in terms of actual project results.

If the team does succeed on the project it is a side effect rather than a deliberate effort on their part. Even then, success would most likely have been even greater with their willful cooperation.

The root of the problem is that coercion is at the heart of hostile relationships between managers and teams. Hostility rarely leads to people pulling in the same direction. Hostility is to the detriment of the project, and to the team, and ultimately to the hostile manager.

In this chapter we will see that there are two kinds of hostile manager: saboteurs, who destroy a team from within, and XXXXX who set up a team for failure from the outside. If we really believe in teams and teamwork, then hostile management isn't going to work.

The alternative to hostile management isn't no management, but rather supportive management, and that comes in two flavors: referees, who make the environment safe for their teams, and leaders, who help teams be effective in their environment. Both are necessary for a team to gel and flourish. When a team is free of energy and morale-sapping battles with hostile managers, and is instead supported by a skilled referee, and an effective leader, the team can focus all its energy on having great ideas, helping the company develop great products, and beating the pants off the competition.

Teams

What is a Team?

Everybody talks about teams and teamwork. We hear constantly that teamwork is meant to be a good thing, so it is no surprise that companies promote it and advertise for “team players”. This would be great if it meant companies sought out and cultivated people who worked well together. In my experience, though, “team players” is often a euphemism for people who will follow orders handed down to them by their boss. Here, “team players” has nothing to do with teams and teamwork, and more to do with obedience.

A manager can certainly assemble a group of people and tell them to work as a team, but those people won't actually function as a team until they have actually become a team. For that to happen they need

to:

1. Unite internally
2. Gel around shared values
3. Flourish in their environment

How Does a Team Unite?

A team starts out as little more than a collection of individuals, each primarily concerned with their own interests. For them to unite as a team there needs to be mutual recognition and acceptance between the individuals that they belong together.

In my experience, a manager can't impose this sense of belonging on the team, the team members have to recognize it in each other, and for this they need the opportunity to get to know one another and find some shared identity that helps them distinguish themselves from their environment. If managers keep team members apart, and deprive them of opportunities to develop a shared sense of belonging, they will never develop a common identity, and the team will not unite. Likewise, deliberately breaking up a team at the end of a project and returning its members to a "resource pool" will destroy any team unity that had started to form.

> A colleague, Neil Craven, sat in a large open-plan office, surrounded by people with whom he never worked. The open-plan space both cut costs on office space and served as a panopticon. It had nothing to do with teams. The company advocated "virtual teams", wherein temporary groups of people were allocated to projects based on near-random availability from a large and geographically-distributed pool.
>
> The company talked up the benefits of teamwork, but as Neil pointed out, people were isolated and lonely. More often than not team members never saw one another face to face, and many would rarely work together again. There was no sense of belonging and nothing to unite the team, other than a short term assignment on a project governed by a common project manager.

How Does a Team Gel?

Once a team has a mutually recognized identity, they need to learn to work together as a team. Well intentioned management efforts often fail miserably in stimulating this. One-off team-building exercises tend to bring short term fun but no lasting impact. Motivational posters are generally seen as

clichéd decorations.

Team building doesn't come by external action, but by the team building themselves. Teams build themselves around shared beliefs about how they should work together. There can be an internal struggle of competing beliefs (some people call this “team storming”) before any dominate, or more likely the founder of the team will impose their own beliefs on the team.

Once a team settles on an initial set of shared beliefs, they begin to form cohesive bonds which help the team gel. This cohesion is important since it compels the team to survive.

I have seen a number a few teams without shared beliefs and as a result there was little or no cohesion among the team members. This stopped them from forming bonding relationships with one another and as a result these teams didn't have sufficient internal strength to last.

> A large financial software company had several teams each deployed in a different country. Senior management was concerned that the teams were isolated, and would benefit from the team leaders meeting once a month to learn from one another. Since I was the head of one such team, I was invited to attend.
>
> The first meeting was interesting, with team leaders flying in from around the world, giving presentations on their various projects. We learned that the projects were very diverse. Indeed, each project required a very different set of beliefs about how to measure success. One project saw success as establishing a foothold in an emerging market. Another saw success as extending well-paid consultancy work with a single customer for as long as possible. A third saw success as delivering a new product funded by a cluster of banks.
>
> By the second meeting it was clear that although we all worked for the same company we had little in common around which to establish any shared beliefs.
>
> There was no third meeting. We all went back to our separate projects and rarely saw each other again.

When a team that has gelled detects a threat to its internal cohesion, the team members experience great anxiety. The need to lessen this anxiety, forces them to do whatever it takes to enable the team to survive. I have seen several teams eject individual team members who threatened team cohesion.

> I headed up a team at Objectware, developing a database management system. One of the team's uniting values was the assumption that if people are shown trust they will live up to that trust and do a good job. One of the team members, Michael, apparently had different

values, including a belief that misdeeds are acceptable so long as the damage is minimal. This caused friction within the team, yet Michael brushed aside criticisms as overreaction from other team members.

One day Michael missed a critical appointment, and it had embarrassing repercussions for the whole team. I tracked Michael down at home, and he made a convoluted excuse which included blaming two other team members. A quick check showed those two team members to be blameless. The team discussed the issue, and voted to have Michael removed. His values were damaging to the team, and his behaviour undermined the values around which the team had gelled. Michael was ejected from the team and soon after from the whole organization.

In more extreme cases, I have seen teams go as far as rejecting their manager or even rejecting the whole organization!

A software company with which I worked had cultivated a highly effective programming team whose products had helped the company prosper. Alas, after several years of success, a couple of expensive sales mishaps led the company to a financial crisis. The fear of redundancy hung in the air. Several programming team members started to look for work elsewhere, and this threatened the unity of the team. I was astonished to see what happened next: The team members held a crisis meeting, after which they resigned en-mass. The whole team relocated intact to a new organization. They had to, in order for the team to survive.

How Does a Team Flourish?

Once a team has united around a shared identity and gelled around common beliefs the team needs to experience repeated successes for the team to flourish in its environment.

It is an obvious but often overlooked fact that a team doesn't just have a relationship with their manager. Anybody outside the team is part of the environment in which the team exists. That environment can be supportive or hostile to the team, or a mix of both. For the environment to be supportive, the team must be useful to that environment, otherwise the team will have no value and may even be seen as a threat to the environment and will be under external pressure to disband.

Success for a team, then, means satisfying environmental needs through their daily work. If the team

members sees that the team's beliefs increase their effectiveness in their daily work and improve the team's relationship with the environment, those beliefs will be reinforced more deeply within the team. Repeated success leads to the team's shared beliefs becoming their deeply held values, and these strengthen the team's internal integration, solidify the team's place in its environment, and prevent the team from fragmenting.

If, however, values prove ineffective, and weaken external relationships, the team will feel threatened. The resultant anxiety will force the team to re-evaluate their values, and if the threat is significant enough, overhaul them with beliefs (leading to new values) that lessen team anxiety and increase the ability of the team to thrive.

Of course, any change in the environment can flip a supportive environment into a hostile threat that may cause the team to perish. Consequently, the team must continually adapt its beliefs (and hence its values) to satisfy its environment's changing needs in order for the team to survive.

NEED AN EXAMPLE HERE

Managers

What is a Manager?

A manger is somebody who has authority over a team. A manager can use their authority to impose their own values on a team. However, that team will judge those imposed values and the manager over time in their effectiveness at enabling the team to achieve its tasks and maintain good internal and external relationships. If the team is successful in its work, then the team will accept the manager's values and recognize the manager as the leader of the team. If the team is unsuccessful in their work, the team will feel anxious about their survival, will reject the manager as their leader, will unite around beliefs and values that they consider more effective, and will find a leader themselves (usually from within their own ranks).

Naturally, a dominating and authoritarian manager will not give up their power lightly and will likely use all their authority to regain control coercive. If the manager wins the power struggle, it can be a false victory. By winning a series of battles, the manager can damage severely the relationship with those on whom they rely the most – their team - and so can end up losing the war. When a manager has established a hostile relationship with their team, authoritarian power can be used create obedience, but it cannot create loyalty.

The Four Manager/Team Relationships

The nature of the relationship between a manager and a team, then, is dependent upon the level of alignment of their values and the fitness of those values to the survival of the team. This means that not all management relationships are healthy from the perspective of the team.

More specifically, since a team has internal and external relationships, and each of those relationships is either supportive to or hostile to the team, a manager can have one of four types of relationship with a team, as depicted in the quadrants of the following grid:

	Internal	External
Supportive	**Manager-As-Leader** The manager is an integral member of the team, who helps the team identify and unite around shared values and adapt and thrive as the environment changes	**Manager-As-Referee** The manager is part of the environment to which the team contributes and supports and strengthens the relationship between the team and their environment by ensuring everybody plays fair.
Hostile	**Manager-As-Saboteur** The manager undermines the team's values, and damages the team's internal cohesion, preventing the team from uniting and impairing their ability to adapt to their changing environment and hence their ability to thrive	**Manager-As-Competitor** The manager prevents the team from forming good relationships with its environment, and therefore increases external pressure on the team to disband

Hostile Manager Relationships

Hostiles are Beneficial

Not surprisingly, the supportive relationships are beneficial to a team: A manager-as-leader can help the team work out how to unite and flourish. Likewise, a manager-as-referee is beneficial since it protects the team from external threats.

Perhaps surprisingly, a team also benefits from people filling the hostile ones. Hostiles provide the team with enemies, and enemies help the team unite and focus their efforts. A saboteur gives the team someone to rally against, thus strengthening the team's internal bonds. A competitor prevents the team becoming complacent, and to force them to be creative and adaptive in order to survive.

Hostile Managers are Damaging

What is not beneficial, though, is for the manager to take on either of the hostile roles themselves, since

the manager will then become the team's enemy. The attitude of many hostile managers is “the boss knows best” and “the employees need me more than I need them”. Whether the manager is right or not, that attitude leads to a confrontational relationship with the team, where the manager has to rely on coercion to ensure obedience.

When it comes right down to it, hostile management is based on mistrust. Managers don’t trust employees to make good decisions, and they don’t trust them to work hard. So, the managers make all the decisions, command the workers to implement those decisions, and put in place controls to make sure the workers aren’t goofing off.

Mistrust means suspicion. If people see that you have little confidence in them, and that you are treating them with suspicion, the relationship between you and them will be damaged. This rubs off on employees, who grow suspicious of your own intentions, losing any confidence they had in you, and they soon feel no loyalty towards you. The results will be mediocre at best, since mutual mistrust drives up costs, saps morale, reduces the quality of decision making, and ensures that team members show little or no creativity.

> Bjoern and I were both directors at a large dot com. A vice president fired people from Bjoern's team as a cost cutting measure, then piled plenty more work on the team. Bjoern complained that his team could not possibly succeed, and was ignored. So, he called upon the highest levels of management for practical help. The response could have been straight out of Dilbert: “I have one piece of advice for you and your team Bjoern: worker smarter not harder!”
>
> Nor surprisingly, Bjoern's motivation fell through the floor. A few months later, despairing that this relationship with management was hopeless, Bjoern left the company

All in all, managers should certainly encourage the existence of saboteurs and competitors, ideally identifying or even provoking such enemies, but manager must shun those roles personally, seeking to instead become either a leader of a referee. That is, cultivating a supportive rather than a hostile relationship. A manager in a supportive relationship with a team will get their full and wilful cooperation.

Manager-As-Saboteur

This is a hostile internal relationship between a manager and a team.

A manager is a saboteur if he or she forces the team to act or to structure themselves in ways that seem good in theory but are actually destructive in practice. Typically, this means forcing on the team values, behaviours, and structures, that continually undermine the team's own values and damage the team's internal cohesion.

In extreme cases this may well be a deliberate act of sabotage, but more often the saboteur has good intentions and is actually trying to help the team be effective. The unwitting saboteur does not realize that their impositions are actually hindering the effectiveness of the team and even threatening the team's existence. From the team's perspective, the saboteur manager appears to be a dictatorial tyrant whose impractical impositions will destroy them.

There are two classic classic cases of manager-as-saboteur: micro-management and bureaucracy.

1. Micro-management is where the manager worries about people making mistakes and doing a less than perfect job. They are afraid of losing control of a project, and having little impact on its success. They are incapable of delegating, and insist on making all decisions themselves. So, they breathe down people's necks, inspecting and controlling them the whole way. This, of course, slows decision making down, and impairs creativity. If you want to see an extreme example of this, search the Internet for memos written by Edward Mike Davis of the Tiger Oil Company.
2. Bureaucratic management puts complete faith in an authoritarian hierarchy and enforces standardized processes with strict administrative rules. Team members are required to be slavish rule-followers and paperwork fillers. Project success is measured primarily in terms of how closely managers and their teams follow the rules. Overall, this style of management sees tidy administration as success in itself, rather than as something supporting the creation of high quality, well designed software. Bureaucracy increases, and project quality decreases. Nobody asks the question that should be on the lips of every manager: "If we were not already using this hierarchy and process, then would we recreate it today and recommend it as the best way for us to start working?"

NEEDS ONE OR TWO EXAMPLES

A team faced with a saboteur manager has only three choices

1. Push back against the manager in order to protect the team's cohesion. This, though, will likely be seen by the manager as unwarranted insubordination and disloyalty, or at the very least as

stubbornness and ingratitude. This often results in some form of punishment by the manager to ensure greater compliance in the future.

2. Give up hope of cohesion and do as the manager demands, resulting in the team being ineffective and fragile and the manager being dismayed at the lack of results
3. Allow the team to disintegrate, thus freeing the team members to join other (hopefully more cohesive) teams, and forcing the manager out of his or her saboteur status.

> Jeremy managed a product team at one of my client sites. He was smart and energetic, but he also set his team up for failure. Jeremy believed that all customers were stupid, and teams should never listen to what customers asked for, because it was invariably wrong.
>
> Jeremy demanded that each member of his team left every decision to him, and had him approve, and invariably change, all incoming customer requirements. Since the team was committed to satisfying customers, Jeremy's relationship with the team was destructive to the team's cohesion.
>
> Members of the team began working strange hours so they could avoid him. They were delighted when he went on vacation or was sick, because it gave them a chance to work closely together and to focus on keeping customers happy. Alas, upon Jeremy's return he invariably reprimanded everybody, forced them to redo their work, and assigned them to separate projects to prevent the conspiracy.
>
> Senior management respected Jeremy for his hard work and dedication, but failed repeatedly to prevent him from sabotaging his team. Since the team was considered more important than any one person, Jeremy was asked to leave. This delighted the team, but left Jeremy certain that everything would collapse without him there to hold it all together. Things did not collapse, in fact the company brought in a new manager, Brian, who established a much more healthy leadership relationship with the team, and helped them to gel and increase their effectiveness considerably.

Manager-As -Competitor

This is a hostile external relationship between a manager and a team.

A manager who is in a competitor relationship with the team doesn't tell them how to do their work, but prevents the team from forming good relationships with their environment. An alarming instance of

this is where the manager seeks to hoard all credit for the team's success, whilst pointing the finger for all failures directly at the team.

> I once joked with a team that my role as their manager was easy: take credit personally for all their success ("look how well they have done under my great leadership!"), and blame all failures on them ("what an incompetent bunch of layabouts!"). After a moment of polite laughter, one of the team members observed: "but that is exactly how my last manager worked!"

More commonly, this hostility is unwitting, where the manager attempts to "protect" the team and the environment from one another – and acts as their intermediary - believing that neither is well suited to interacting directly with the other. This prevents goals from being clearly understood, limits opportunities for direct feedback, and slows down all communication. A typical trait is where the manager continually over-commits to customers on what the team can achieve in a give time-frame, resulting in oscillation between optimism and disappointment. This isn't beneficial to either the team or the customer, and damages the relationship between them.

> For a short while, I lead a few teams for a recruitment company, which must remain nameless to protect the innocent and the guilty. One of my teams complained that my own boss had just piled a huge amount of work on them, with an impossible deadline, and on a project they knew would be cancelled anyway. I turned to my boss to explain that he was setting the team up for failure, and therefore damage our relationship with the customer. His response was distressing: "Your opinion is irrelevant. This is a hierarchy not a democracy. We have already made these commitments to the customer, and your job is to deliver on them." Not surprisingly, I and members of that project team chose to leave the company soon after.

A similar trait is where the manager keeps changing their mind about priorities, without the team ever having the chance to complete anything. Thus, little is ever actually delivered to customers, and the team loses all credibility.

> NEEDS AN EXAMPLE

A competitor relationship will be seen as a threat to the existence of the team and so will usually meet considerable resistance from the team. In this type of relationship, a manager can always use authority to quieten the team and demand their obedience, but the manager will never have their willing

commitment and cooperation. The problem is that a competitor relationship is fundamentally a dysfunctional relationship between the manager and the team that will lead to under achievement, considerable frustration, and often to a break down of the relationship and the break up of the team.

Supportive Manager Relationships

Trust

When it comes right down to it, a hostile relationship is never going to bring out the best in people. You may get their compliance, but you won't get their willful cooperation. Supportive management, on the other hand, means establishing relationships based on trust rather than on authority. Sure, trusting people can be risky, but not trusting them turns out to be even riskier. When trust goes down managers and teams learn to hoard information, manipulate facts, cover up mistakes, seek personal credit for successes and pass blame for failures on to one another. As a result, speed goes down, cost goes up, and effectiveness goes out of the window. In his book, The Speed of Trust, Stephen M.R. Covey calls this "the low trust tax" compared to "high trust dividends".

> When I first joined a major Silicon Valley consultancy firm, I was assigned to a team of around a dozen people, under the leadership of Greg. We were told by Greg that he was going to be a hands-off manager, and would be protecting us from outside threats and helping us build and maintain good relationships with our client.
>
> That, at least, was the theory. In practice, Greg was under immense pressure from his own managers to reduce costs and increase revenue. Consultants were now viewed as little more than an expense – despite the fact that the team generated considerable profits for the firm.
>
> Things reached crisis point when Greg announced that the company was shifting from consultancy to packaged software – since software "is like printing money – it doesn't require a salary". Consultants were ordered to use every opportunity to sell more software licenses to clients so that the company could eventually prosper from license revenues alone and rid itself of its consultants. Not surprisingly, this was seen by consultants as considerable hostility from managers, who were now perceived as enemies rather than allies. A number of the best consultants left, and many of those that remained lost trust and became rather unmotivated.

Weekly meetings with Greg tried to rebuild confidence. After several weeks of continual reassurance that the company was backtracking, and now valued consultants highly again, trust started to be re-established, and motivation began to return. Yet, a short time later, the company announced it had sold a large part of the consultancy business, leaving senior managers richer and the remaining consultants embittered.

Trust needs to be two-way. When managers learn to trust their teams, and – more importantly – teams learn to trust their managers, those managers no longer need rely on coercion, and the mediocre results that stem from it, and can instead cultivate loyalty and true commitment. Under these conditions, team members become more motivated and more willingly yield their valuable knowledge and creativity to their projects and teams, and hence to the company. In short, trust helps get the best out of people; As well as keeping people happy, trust keeps costs down and value up: delivering more for less.

Before the team can gain trust in you as a manager you have to demonstrate to the team that you are trustable. Trust means confidence. Being open and honest is a good start, but just being nice to people isn't enough. That might help the team to like you, but it won't inspire their confidence in your competence in your work and your ability to manage them.

Early in my management career, I tried my hardest to make my team like me. I wanted to be their friend. They were always happy to go for beers with me, but I never really earned their trust. It took me a few years to realize that being nice isn't the same as being supportive.

My excuse was that own boss, a bit of a tyrant, would often make foolish demands. I rarely pushed back since, to be honest, I was afraid of him. Instead, I would approach my team with a smile and say, "Hey guys, we have been told we have to do this. I know it doesn't make much sense, but then we all know the big boss is an idiot, so just get on with it the best you can."

This as a terrible management style. It left my team powerless and unsupported. What I failed to do was to care about the team. I was a weak manager: I cared only about myself.

Earning confidence involves demonstrating your personal credibility by focusing on mutual benefit, and following through on your commitments to the team. That means being consistently outstanding in your job, and able to help the team become outstanding in theirs.

Above all, you have to accept personally responsibility for results. Results are vital here, since others will continually evaluate your credibility based on your past performance, present performance, and anticipated future performance. Rather than taking credit for successes and passing blame on to the team for failures, you need to learn to share success with the team and accept blame personally. Let results, rather than title, speak for themselves.

> Philip was an extremely talented software team leader at Selima Software in the UK. After several years earning his spurs, he was promoted to a directorship, and ultimately was made a partner in the company.
>
> Philip had previously been very much respected for his expertise, but unfortunately, the new title went to his head. With his new authority, he increasingly relied less on expertise and more on command and control. His style became extremely dictatorial, and his willingness to help out the guys in the trenches evaporated.
>
> Gradually, Philip's team lost their respect for him. He found himself increasingly isolated, so that ten years down the line he was reduced to administrative task such as approving people's monthly expenses. Even in that role he was renowned for his pettiness: spending half a day chasing somebody who had accidentally expensed a chocolate bar along with travel costs.
>
> Eventually, the level of mutual trust had slipped so far, that Philip's presence was doing more harm than good. The other partners voted him out of the company. Philip's reaction? “But I am a partner! You can't throw me out. I decide who can stay and who goes!”

Once you have established that you are trustable personally, you can begin to build trusting relationships by consistently behaving in trust-building ways. At first, these will likely be one-on-one relationships with individual people in the team. As your trust grows, these relationships can gel into team-wide trust for you as a manager

> NEED AN EXAMPLE.

In short, if you want great results, and fast, you have to stop clinging to suspicion, relying on underhand tricks or treating people as if they were expendable, and start building, maintaining, and above all living up to trusting relationships with your team. The manager is then no longer the person the team is frightened of, but rather the person who helps the team achieve great results.

Manager-As-Referee

This is a supportive external relationship between a manager and a team.

In the late 1980s and throughout the 1990s, several books and articles claimed that the most effective management approach was:

- Hire the best people you possible can
- Give them very clear goals
- Get out of the way while they deliver great results

This was clearly a reaction to overbearing, hostile, management which slowed projects down and impaired innovation, and this lead to one of the hot management buzzwords of the time, which was “empowerment”. This type of relationship is based on and establishes trust. In contrast to hostile relationships, which are generally based on suspicion.

The manager and the team negotiate what they can reasonably expect from each other and then trust each other to get on with it. If each side then satisfies those expectations to an acceptable degree, the level of trust in the relationship is increased, and the relationship will then require even less formality.

Empowerment is meant to cut through unhelpful management interference, and give a team more power to decide how to be effective in their work. The underlying idea is that a team will by necessity find its own means of being effective in its environment. The team will change its own beliefs and values, and hence its internal structure and its ways of working, in order to survive. The manager resists the temptation to use authority to impose their own beliefs and values on the team and interfere with the internal workings or structure of the team. Instead, the managers remains sensitive to the team's need and ability to unite and flourish for themselves. A manager that cannot resist interfering in this process, no matter how well intentioned, risks becoming a saboteur.

This sounds great – since it frees managers of a great many responsibilities and frees teams from unwanted interference. However, in practice it often has meant giving teams lots of responsibility with little management support when they needed help.

This lack of practical support may have been due to a belief that having no management was better than having hostile management. It leaves many empowered teams feeling stranded; facing obstacles they lacks the power to tackle themselves, and with little or no ability to shift goals and re-steer the project

as the environment change over time.

> I experienced empowerment first-hand when I headed up a project team at Objectware in the Sheffield, England. The CEO, Richard Jowitt, told us to replicate a complex and competing product. Things began well, but over time we began to hear rumours that an increasing number of customers thought our product was a "me-too" offering, with nothing innovative to tempt them to purchase it. Whenever we went to Richard for guidance on this issue he always replied "just get on with it, let me know when it is finished, and then I will get on with selling it". After three years of isolation, we delivered a perfect copy of the competing product. By the time the product shipped, though, few customers were interested, and even fewer were willing to buy it.

Sure, teams don't benefit from hostile management interference, but teams will face obstacles they cannot tackle themselves due to lack of power to do so, and for this they need help from managers who do have the necessary power. This, then, is the manager-as-referee relationship. It is a more responsible form of empowerment, where the manager follows all of the bullet-pointed guidelines above, but also focuses on making the environment safe for the team. Here, the manager uses their authority and political clout to ensure that those outside the team play fair, to give the team every chance of succeeding. For example, tackling bureaucracy, boosting insufficient budgets, and repelling stubborn competitors and saboteurs. That is, rather that directing their authority inwards to command and control the team, the manager-as-referee directs their power outwards to support the team, removing obstacles that the team is powerless to address themselves.

> Alfa, a Luxembourg-based group of companies, hired me to manage some of their project teams. The principle at Alfa is that the manager isn't the superior, and the team members his or her subordinates. The manager isn't there to be feared by the team but respected and trusted by them. Alfa views the job of management as removing obstacles that get in the way of project teams, and continually helping those teams to become more effective in doing their work.
>
> When I was with Alfa, I noticed that many of Alfa's customers actually had very large software departments of their own, yet their teams floundered repeatedly on a variety of projects, and they repeatedly brought in Alfa to rescue them. Each time I looked into this, I noticed that those departments were dominated by hostile management relationships. In my

opinion, the reason Alfa succeeded was not that they had smarter people than their clients, but that they had a more effective relationship between management and teams – that is, manager-as-referee. By keeping hostile managers at bay, and eliminating impediments such as non-contributory bureaucratic work, Alfa's managers helped teams get more done faster and with fewer people.

Manager-as-Leader

Manager is a title given from on high, and confers official authority over a team. Leader, on the other hand, is a position given by the team. It reflects their trust that the leader has the ability and the intention to help the team flourish.

A leader cannot be imposed on a team. If senior managers do appoint a team leader, that leader must still earn their leadership relationship with the team. A leader who proves ineffective will be rejected by the team, and management interference in this process risks making that leader a saboteur.

A leader helps the team from within, rather than imposing on the team from without. Leadership means helping the team find shared beliefs around which to gel, and helping the team succeed repeatedly in their work. Success is vital here. A leader who simply helps the team do whatever they want isn't providing any leadership at all. The leader has to help the team become self-critical, and constantly monitor and reevaluate, and where necessary completely overhaul their values, in order to deliver results that matter to the environment. Unless the environment is satisfied, the pressure on the team to disband will increase. When a leader helps a team find ongoing success, the environment will continue to support that team, and hence team will flourish.

The most effective method I have seen to foster leadership is for upper management to introduce a career path that encourages it, rather than forcing all seniority into a professional management hierarchy, with upwards reporting and downwards control. Leadership doesn't mean having more control over teams, it means having more responsibility to them. When there is only a professional management career path, people's only means of advancement is up the management hierarchy, so their temptation is to focus on impressing their own managers by showing that they are good management material themselves. As a result, they can lose track of their commitment to helping the team as a

whole develop great products and be pulled toward selfish individualism, which is good for themselves personally but harmful to the overall effectiveness of the team.

> Many years ago I work for AT&T in the research labs. It was explained to me that in many organizations only the most junior staff do actual research and product development work, since anybody with seniority is faced with two choices: stagnate, or move into professional management. Such companies are starved of deep expertise, since they have no career paths that allow it to develop. As a result they tend to have mediocre products, and often have to have to resort to expensive and fleeting external consultants for inspiration and help.
>
> In contrast, the leader of our research group at AT&T was a senior consulting engineer, and later a research fellow. These positions recognized his deep research background and abilities, and were equal in rank and salary to senior management positions. Rather than being bogged down in bureaucracy, such as cost accountancy, this leadership career path let experienced people spend the majority of their time improving the quality of research.
>
> That is, at AT&T the best people were allowed pursue a leadership career path and keep doing great research, rather than being forced into professional management. With the best people doing research rather than administration, it is no surprise that AT&T's research labs were world class.

Power

Hostile managers rely on hard power – authority, reward, and punishment - to coerce a team, since that is the only power they have over that team. A manager-as-referee also relies on hard power, but this time to control the environment on behalf of the team. A manager-as-leader, though, does not rely on authority, reward, and punishment at all, but rather relies on softer forms of power that help the team control itself.

The differences between hard and soft power can be seen in the five forms organizational power, identified in 1954 by two social psychologists, French and Raven.

The first three forms of power are hard power. They are used extensively by managers in a command and control hierarchy. Hard power is based on the idea that employees have to obey orders handed

down to them from the manager, who controls their pay and their promotion prospects:

- **Legitimate Power**: This is where status and authority come from your title. Teams have to do things because their boss tells them to: The manger gives commands, and subordinates obey.
- **Reward Power**: Here the manager offers a reward that the team wants or needs. The manager makes demands, and the team members comply to increases their chances of a bonus, a pay increase, or a promotion
- **Coercive Power**: The manager gives commands, and team members are punished for disobedience. The managers threatens people with a lower status, a salary freeze, or even losing their job unless they do as they are told.

Hard power still has tremendous use for *supporting* teams, by making their environment safe, as described for the manager-as-referee relationship. However, the domination strength of hard power for *controlling* teams has reduced dramatically over the past few decades.

There is now much less reliance on physical labor, and menial work, for which control by hard power was well suited. The primary means of production no longer lies in ownership of factories or land, but in the creative minds of the employees. In the computer software business in particular, the majority of work involves intellectual effort, particularly deep technical knowledge, high levels of creativity, and an ability to lean and adapt quickly. It is entirely up to each employee to decide how much of their mental ability they will put into their work.

If these knowledge workers decide to leave the company, they take that means of production with them. This makes highly capable people a far greater asset than ever before. Companies are in fierce competition with one another to attract and retain the best and the brightest. As a result, in many cases, the attitude of employees has shifted to “the employers need us more than we need them” and often they are right.

This can be a bitter pill for many authoritarian managers to swallow. Nevertheless, managers cannot use authority to force a team of knowledge workers to work harder, and certainly not to be highly innovative. Commands backed by threats will bring only begrudging compliance; they cannot foster enthusiastic cooperation, creativity, and loyalty - all of which are essential to gaining an edge on your competitors. Because of this, the balance of power has shifted more towards much softer power, and these form the basis of leadership:

- **Expert Power**: Those most skilled to solve a specific problem call the shots to fix that problem. Hierarchy is irrelevant here. The professional manager is often the least knowledgeable person about how to best address a given issue, and has to defer to the expert knowledge of the team. A leader is more likely than an authoritarian manager to know how to do the work of the people they lead and may well have come up "through the ranks". A leader understands the work that the team does, notices when a team is in trouble, can jump in and help them in difficult times, and can train and guide them in their work. The leader continually strives to expand their knowledge and develop their abilities, both in the work of the team and in leadership of that team.

- **Referent Power**: Rather than respect being demanded from the manager based on their title, it is earned by people through merit. A leader who is an inspiring role model and shares and respects the teams values will gain their loyalty and their best cooperation. The team will know that the leader has their best interests at heart, and the team won't want to let that leader down.

Balancing Power

In summary, when teams say "we need less management and more leadership", what they usually mean is that hostile management relationships may well strengthen managers personally but they weaken the organization as a whole.

Calls for more leadership aren't about eliminating management, but about supporting teams rather than stifling them. This needs two supportive management roles, manager-as-referee and manager-as-leader, which together balance authority with other forms of power. Relying on only leadership risks threats from the environment which the softer powers of leadership are insufficient to repel. Relying only on refereeing risks leaving a team without the ability to gel, and therefore without the internal cohesion necessary to work as an effective team. Together, the leader and the referee can help a team gain sufficient internal strength and external safety to thrive and do great work as a team.

Alas, in an organization heavily populated with authoritarian, command-and-control managers, shifting from dictatorship to the balanced power of refereeing and leadership is going to be a serious practical challenge. In a number of companies, senior executives have asked me: "We agree we need to do this, but in practical terms what are we going to do with all the managers we have in place now?".

There is no denying the company is going to meet a lot of resistance from authoritarian managers. As

they start to feel threatened, they will try to hold on to, or even increase their power within their organization.

Teams and leaders can't be expected to fight it out for themselves, because soft power will always lose in a political battle with authoritarian hard power. The only way I have seen this work is for upper management to step and use their own authority to cut through power struggles and commit to making this change happen.

At the same time, senior executives must continually reassure professional managers that leaders do not undermine their power, rather they are parallel to it. Some may welcome this, since it brings new opportunities for capable people; particularly those felt they were forced unwillingly into management as their only career path, always regretted leaving behind their technical expertise. Occasionally, I have seen such managers switch over to a leadership career themselves, and in most cases with good results since they now have learned the ways of management as well as drawing on their own prior expertise. Those with the recognized ability to lead – rather than dictate – can then have the whole power of their team behind them.

Those who decide to remain as professional managers rather than leaders, may wish take on the manager-as-referee role. This does, of course, reduce their direct authority over project teams, and instead focus them on supporting teams and leaders, using their political clout to eliminate obstacles that impede team productivity. Although this may be uncomfortable for some, it does allows managers who lack leadership ability to still retain their management status and make valuable contributions to the company.

In cases where managers are unwilling or unable to become either leaders or referees, it is vital to recognize that so far we have focused only on the power relationships between managers and teams. There are other vital management roles that are unrelated to such power relationships; particularly administrative management, such as budgeting and cost accounting, to which many managers seem drawn as a career path.

Section 2

The Production Line Metaphor

Managing software projects can be a pretty scary business. All sorts of things can go wrong. Too many projects have ended up over budget, or late, or producing something that was poor quality or that nobody really wanted. This makes managers of software projects understandably twitchy, and it is no surprise that many have turned to proven project management techniques for help.

Chapter 5 has you working as the production manager in a chocolate factory. You will see first hand the efficiency of a well run production line and face the challenge of ensuring that the factory continues to run smoothly.

Chapter 6 traces the history of production lines, and shows how their success in the mass manufacturing led project managers in other industries to employ production-line principles to tame their own project risks.

Chapter 7 notes that many software project managers have been inspired by the success of production lines in other industries. They have worked hard to reduce project risks and inject greater predictability, reliability, and discipline, by creating software factories with simulated production lines monitored and controlled via proven project management principles.

Chapter 8 explores several software development methodologies through the perspective of the production line metaphor, with the aim of helping you gain not just an intellectual understanding but also an emotional feel for them.

Chapter 5 : The Chocolate Factory

Your Life as a Production Manager

Think of a production line. Not in the context of software development yet, since that will stir your own emotional reactions to the suitability of production lines for software projects.

Instead, think of a production line in a factory. Let's say they are making boxes of orange-cream-filled chocolates. Pause to visualize the factory in your mind's eye.

Start with an image of a clean and well-maintained factory, producing boxes of chocolates reliably and efficiently. Look around the image. Notice the production line, with chocolates laid out at various stages of production along the conveyor belt. Look at the machines at various points along the production line, and see the people operating them.

Take a moment to run the production line like a movie in your mind. Visualize liquid chocolate and orange-cream pouring into large containers. Add colour, and smells and sounds if you can. Notice what happens as the first machine creates empty chocolate shells and lays them out on the conveyor belt. See the chocolate shells pass smoothly along, as next operator presses a button and pulls a lever, while their machine fills the chocolate shells with just the right amount orange cream. Observe the next machine pick up the filled chocolates ten at a time, and put them neatly into boxes. See the chocolates boxes automatically stacked and packed at the end of production line, ready to ship out to customers.

Now imagine you are the production manager. Visualize yourself walking up and down, clipboard in hand, feeling the satisfaction of the production line running smoothly, reliably, and repeatedly. Hour after hour, day after day. Everything is running like clockwork.

Yikes! One of the machines on the production line breaks down. Half-filled chocolates are piling up along the conveyor belt. Workers are panicking: liquid chocolate, orange cream, and chocolate shells are spilling off the conveyor belt, and onto the factory floor. Feel your heart racing. Feel the panic setting in. This is your responsibility. You have to sort it out. What are you going to do about it?

You have faced these situations before, and know how to deal with them. You run to a big red button on the factory wall, slam it with your open hand, and bring the production line to a halt. You shout out orders, you put into effect standard procedures to clean up, find the source of the break down, fix it, and start the production line running smoothly again. Within an hour, production is back to normal. Panic over.

The Luxury End of the Market

It is six months later. You have a well-earned reputation as a great production manager. You are head hunted by a chocolate factory that specializes in a luxury range of hand-made chocolates.

Picture in your mind half a dozen artists sitting around a table making luxury chocolates. There are splashes of chocolate and smudges of orange cream on the floor and on the table. The artists swap stories about their families. They smile. They laugh. It seems to be a fun place to work. Hear the sounds, smell the smells, feel the atmosphere.

One of the artists finishes shaping a chocolate shell by hand, and reaches over to the bowl of orange cream. There isn't much left. She gets up and makes a joke, the others laugh. She walks over to the refrigerator, takes out a fresh bowl of cream paste, and takes it back to the table. There isn't any orange oil left either, so she heads off to a small storage room, and comes back a couple of minutes later with a small bottle. She pours a few drops of orange oil into the cream paste. Oops, a little too much oil this time; this batch is going to taste a more zesty than usual. Better be more careful next time.

With great patience and skill, she slowly blends the orange oil into the paste.

Pause a while to review the scene. As the production manager, you are responsible for how this factory is run. Compare the scene in your mind with the production line that was under your control in your last job. What are you feeling? What are you thinking?

A couple more artists have now finished shaping chocolate shells, and are waiting for the orange cream to be ready. Another artist is adding final decorations to a handful of chocolates. His chocolate knife slips, consigning two more chocolate to the misshapes bin. He puts the surviving chocolates one by one into individual boxes, and ties each one with a ribbon. When has twenty boxes filled, he pick them up, puts them in a bag, and takes them over to the next building. Five minutes later, he is back, making chocolate shells again.

You sample one of the chocolates. They certainly do taste good. But you have a nagging feeling that this factory needs a shake up. You wonder if things shouldn't be run more efficiently. You can see the mistakes being made here. These hand made chocolates cost ten times as much to make as the regular orange-cream filled ones from your last job. There is a lot of waste, and production is slow and unreliable. The work is extremely labor intensive, and subject to the whims of highly-skilled and highly-paid chocolate artists. Things could be run much more smoothly than this.

By the end of the day, the six artists have made four hundred and seventy three chocolates. Less than the five hundred per day target. Production is your responsibility now, and you are held accountable if targets aren't met. Hopefully, the chocolate artists will make up for the shortfall tomorrow.

What images are forming in your mind? How do you feel the way this factory runs?

A couple of days later, the office manager grabs you. A competitor, keen to dominate the luxury end of the chocolate market, has found a way to reduce dependency on artists, by automating much of their work. The competitor employs low-cost machine operators, making high end chocolates on a production line, at a tenth of your cost, with waste reduced by eighty percent, and production volumes twenty times higher than your artists could ever manage.

Visualize that competitor producing those luxury chocolates. See box after box coming off the production line. See yourself tasting their chocolates. They taste every bit as good as yours, and cost a fraction of the price. You see the efficiency of the production line proving itself all over again. You look back to your chocolate artists, swapping jokes, taking four and a half minutes to make a single chocolate, making mistakes, throwing rejects into the misshapes bin.

This is your responsibility now. Senior managers are putting you under intense pressure to make sure production runs smoothly and reliably, and profitably target are met. No excuses. You are either up to the job, or you will be out of a job. What are you going to do?

A hotel chain offers your competitor a contract for eighty thousand luxury chocolates per month. Hotel guests find the chocolates individually-boxed on their bedside tables. They taste them, they enjoy them, and they remember the brand name. Next time they are in the candy store, they see your brand priced much higher than the one they enjoyed so much in the hotel. Their hand reaches out for a box of chocolates. Which are they going to choose?

Can the artists in your factory compete with the efficiencies of the production line? If you can't make this factory competitive, the factory will be out of the luxury chocolate business, and you will be out on the street. What are you going to do?

From Chocolates to Software

A year later, and with plenty of experience as a production manager in chocolate factories under your belt, you decide it is time to move on. You are offered a well paid job as a project manager at a computer software company, and you accept it. You know next to nothing about writing software, but you know exactly how to make things run smoothly and efficiently.

During your first couple of days in the company, you notice far greater chaos than you had expected. Too many projects are missing deadlines and going over budget. It seems to you that these folks haven't got their act together. Well, that's probably a big part of why they hired you in the first place.

The senior executive in charge tells you that you have six months to get projects under control.

As you start to settle in, you see that the company is heavily dependent on the whims of a few highly skilled and highly paid computer programmers. These folks can't see to give you a straight answer about how long something will take. When they do finally deliver something, it often turns out that customers don't want half of the stuff the programmers put into the software. Even worse, the bits the customers do like seem to be pretty unreliable.

This is all starting to look very familiar. The software company is being run rather like that hand made chocolates factory was run before you injected more discipline, predictability, and cost control.

Think back to the feelings you had when you first arrived at the hand made chocolate factory. Remember the artists, sitting at their table, with chocolate and cream slopping onto the floor. Remember how inefficiently they worked. Remember how that competitor nearly put them out of business. Remember how much stress you went through. You learned the hard way that a company heavily dependent on the work of artists can't hope to compete with the efficiencies of a well run production line. You saved that factory from bankruptcy. You turned that business around. You created one of the most efficient production lines in the industry.

It looks like you are facing the same kind of situation all over again. If this company doesn't sort itself out, you can see them going out of business. You know how to deal with this kind of sloppiness, and run things efficiently, smoothly, and profitably. You have done it before, and you can do it again. You've been given six months to make a difference. So, what is your next step?

Chapter 6 : The Production Line

The Great Puritan Migration

To understand the Production Line metaphor we first have to understand the history of production lines, which takes us all the way back to the Great Puritan Migration from Europe to New England between 1630 and 1642,

Leading by Example

Puritans brought with them the Protestant work ethic, which found morality in hard work and collaboration in pursuit of common interests. This required a combination of craftsmanship and organizational skills.

In Europe, craftsmanship and organizational skills tended to be separate: A rich elite ruled over a poor working class in an aloof dictatorship. In stark contrast, American managers knew their trade; In addition to having great organizational abilities, they were craftsmen who led by example; enthusiastically rolling up their sleeves and getting their hands dirty. As a result, they grew close to their workers and their concerns and gained a broad and deep understanding of all aspects of their businesses. This helped them make better informed decisions than their hands-off competitors in Europe.

This American entrepreneurial spirit flourished for three hundred years. Then came the industrial revolution, and with it a decline in the belief that a competent manager required craftsmanship.

Mass Production

The separation of craftsmanship from organizational skills stemmed largely from improvements in mass production in the manufacturing industry.

With craftsmanship, the product stays pretty much stationary, while the craftsman moves around, measuring, changing tools, and switching tasks. This wastes valuable time, slowing the overall rate of work. It also requires a multi-skilled, and hence highly paid, craftsman. These issues keep the price of

finished goods high. And the quality of the final product varies; depending not just on the individual skill of the craftsman, but also on the consistency of their work over time.

With mass production, on the other hand, workers stay stationary as the product moves along a production-line. Work is broken down into a series of stages, where the output from one stage becomes the input to the next. Raw materials are fed into the start of the production-line; a machine operator assigned to each stage performs a specific and partial transformation to the products that passes them; and finished goods fall out at the end.

Production Line Efficiency

The production-line proved far more effective than craftsmanship at producing large quantities of identical goods cheaply, quickly, and reliably. For a start, each worker needs only to be semi-skilled: with a narrowly defined role, performing simple and repeatable operational tasks. This makes workers cheaper. They also work faster, since they remain at fixed stations on the production-line, rather than wasting time constantly switching tools and tasks. Finally, the probability of human errors and variations in quality is reduced dramatically, due to the repeatability of the process and the consistency of machinery.

Individual craftsmen simply could not compete with the efficiency of mass production. This reduced the need for general craftsmanship in both managers and workers. Workers became machine operators on the production-line. Managers focused on the organizational administration of their factories.

Professional Management

Inspired by the success of production-lines in manufacturing, managers in other industries looked for similar control over their own businesses.

Companies brought in scientific management and efficiency experts, and management engineering firms, to help define corporate structures, with fixed roles for individuals, and standard ways of working to ensure repeatable results. Companies published organizational charts, establishing hierarchies of authority and ensuring that all decisions are made or sanctioned by managers. They wrote mandatory procedures manuals, outlining all the work-steps and rules to be followed by workers, and put controls in place to monitor worker obedience. Managers became rule enforcers. They would

“throw the book at people” if they broke the rules.

The result is that managers no longer had to understand, or be competent in, the work of the people on the ground. Instead, they simulated production-lines across all kinds of industries: defining repeatable processes, eliminating dependency on craftsmanship, reducing variations in quality, and maximizing time and cost efficiency. Managers, then, became hands-off business administrators rather than leaders in the traditional hands-on role-model sense.

Universities soon offered education entirely in business administration alone. It was no longer necessary to start on the shop floor, and move up through the ranks. Management had become a career path on its own. This, then, signaled the rise of professional management.

Project Management

It is this professional management style that I learned, first in academia and then through many years on the job as a project manager. Like most managers, I have always wanted my project teams to create great products cheaply and quickly. This isn’t easy, though. If left uncontrolled, projects often go haywire: they go over budget; or drag on for too long; or miss some product requirements; or quality starts to slip.

Project management is about controlling such risks, and production-line management showed the way: Plan things before production begins; Assign workers to tasks on the production line; Monitor production to see if things are going according to plan; Regain control as necessary if they are not.

These steps have come to be regarded as best practices, even standard practices, for project managers across a wide range of industries. Plenty of courses are now available that that show you exactly how to repeat these steps for your own projects. Certification is available that recognizes your professional project management competence.

Plan, Command, Control

Project management is plan-driven management; great faith is put in up-front plans, and projects are driven as close as possible to those plans. It is also sometimes called command-and-control management, because workers are told exactly what to do, and are monitored closely by administrative managers to ensure they do it. Here are the basics, all of which are based on the proven efficiencies of

the production line:

How to Plan:

- Write a detailed specification for the product, including a required quality standard
- Break down, into a multi-stage sequence, the work required to make a product matching that specification
- Define specialized work stations (roles for workers) at each stage
- Identify an optimal sequence of simple and repeatable tasks required for each such work station
- Calculate the time required to complete each task; Produce a corresponding time schedule for each stage and a start and end date for the project as a whole
- Determine the economic resources (raw materials, cost of labor, machines and land, etc) required for each task. Sum these to determine the cost for each stage; Sum these costs to determine the overall project budget

How to Start Production:

- Take workers from the human resource pool, and assign them to work stations identified in the plan
- Order workers to perform repeatedly the tasks associated with their station, at the work pace required to finish the project on time and within budget
- Press the start button

How to Monitor and Control:

- Continually track:
    - The output at each stage, and the finished product at the end, checking that they meet the quality standard
    - The rate of production, checking that the pace is sufficient to remain on schedule
    - Economic resources consumed, checking that the project is still within budget

- Where inspections find problems, regain control:
  - Stop the production-line.
  - Rectify the cause of the problem where possible by:
    - Shouting at the workers
    - Fixing broken equipment
  - Resume production
- Where the cause of the problem cannot be rectified, try the following, in order:
  - Increase the budget
    - Offer economic incentives to workers
    - Add more workers to the production-line
    - Add another production-line
  - Extend the project schedule
  - Change the product specification
  - Scrap the project

Section 3

The Model Building Metaphor

In the last section, the production line metaphor told us that up-front planning helps prevent downstream problems. Get the requirements nailed down, and run the project according to plan, and you eliminate nasty surprises.

The model building metaphor tells us that in practice, it is unrealistic to expect requirements to be nailed down completely up-front. People tend to be a bit unsure about their requirements at first, and even when they do know them, requirements are prone to shift over time.

Expecting everything to be specified down up-front is just going to lead to frustration: you might deliver what the customer asked for but not what they really want.

Requirements have to be given time to settle down. Trying out scale models can help with that, and lead ultimately to a stable software architecture, providing a sturdy base on which to build, yet remaining flexible enough to accommodate future changes.

Section 4

The Customer Satisfaction Metaphor

In the last section, the model building metaphor told us that requirements settle down after a while, leading to stable architectures, with future changes fitting neatly into pre-built slots.

Well, what happens if you come across requirements changes that were unexpected and don't fit into those slots? If the customer really wants those changes, you have a problem. Making excuses that something doesn't fit neatly into the architecture isn't going to help with that.

The customer satisfaction metaphor tells us that the most important concern for a project is ensuring the customer remains satisfied. The architecture isn't there to support the software; it is there to support the customer. If it can't, it hinders them.

Software must remain soft, and its structure needs to change in response to unexpected shifts in customer value. Projects must shift their focus outwards and be driven by the changing needs of the customer, rather looking inwards for stability in the internal structure of the software.

Section 5

Shifting Metaphors

Chapter 13: Corporate Culture

A Job Vacancy

A mid-sized company, Mops and Sponges, has a job vacancy.

Mops and Sponges Inc

seeks a

Software Team Supervisor

- *Team player*
- *Results oriented*
- *Excellent problem solving skills*
- *Works well under pressure*

Three people apply, each believing themselves to be the ideal candidate.

Dear Mops and Sponges,

I believe I am the ideal candidate for the position of Software Team Supervisor.

Team player: I thrive when I work with a gelled team of talented people working closely together, each contributing to all aspects of that project, and each committed to broadening their skills and helping other team members broaden theirs.

Results oriented: I see the main goal of a project as delivering great software that customers care about, rather than succumbing to the distractions of excessive paperwork and time-sapping meetings. I consider myself strongly results oriented: always focusing on getting software into customer hands early and often, with their ongoing feedback steering future effort.

Excellent problem solving skills: I never let conventions and old habits get in the way of success. I am always there whenever projects need radical ideas to tackle difficult issues or when a team needs to completely change the way it works to become far more effective at satisfying changing customer needs.

Works well under pressure: I am well aware that customers are under great pressure themselves, and the team has to do whatever it takes to keep those

customers satisfied. In my experience, the key to handling pressure is adaptability – being continual responsive to unexpected events. That could even completely rewriting the software from release to release if customer's changing expectations require it.

Best Regards
Albert Agile

Dear Mops and Sponges,

I am confident you will find me the ideal candidate for the position of Software Team Supervisor.

Team player: I make sure that everybody in the team contributes towards and commits to a shared understanding of the overall system by building models of that system and using top quality software tools to ensure their consistency. These models form the basis for the software system's architecture, upon which all of the team's remaining project work builds.

Results oriented: I keep everybody focused on early delivery of a reliable software architecture, providing a solid foundation for the rest of the system to flesh out.

Excellent problem solving skills: I have considerable experience of looking at complex situations and quickly seeing within them the most important parts and the essential relationships between them. I work hard to capture these abstractions in a high level architectural design, which I and the team refine as we learn more about the intricacies of the project.

Works well under pressure: I handle pressure by being well prepared for it. Once the software architecture has stabilized, points of potential future change will have been identified and accounted for in that architecture. As new and urgent requirements come in, they can be supported rapidly in the software by making use of those predefined points of architectural variability.

I look forward to hearing from you
Mary Modeler

Dear Mops and Sponges,

I am confident that I am the right person for the Project Team Supervisor position.

Team player: I don't let anyone on the team rock the boat. Everybody needs to remain disciplined and follow established best practices for projects to succeed

smoothly. I make sure each person understands their responsibilities well and follows through on all tasks assigned to them.

Results oriented: When senior managers makes decisions, I work hard to ensure people execute on them. I provide regular status reports to managers, so they can track project progress, and remain informed about any deviations from plan which require their intervention.

Excellent problem solving skills: I have considerable experience with effective project planning: breaking a complex work effort down into a well thought out sequence of small simple tasks, with clearly defined roles for workers, ensuring that all work pieces are completed on time, and all parts are ready for final assembly in budget and on schedule.

Works well under pressure: When a project veers off plan, I don't panic; I tighten the screws, hold people accountable for their mistakes, ensure the process and plan are reinforced, and bring the team and the project back into line.

Yours sincerely
Freddie Factory

Hiring the Best Candidate

Based on the information in these letters, which of the three candidates looks like the best fit for the vacancy? Don't ask yourself which candidate you would hire personally. Ask yourself which candidate is the best fit for Mops and Sponges.

Of course, it is impossible to answer that question without knowing something about the culture at Mops and Sponges. Somebody who doesn't fit in with the corporate culture will have a hard time working well with their colleagues, and probably won't hang around for long. Sure, its important to see if candidates are technically capable of doing the job, but it is at least as important to see if they will gel well with their colleagues.

Self-Serve asked me to build up one of their teams. The team had five open positions, and received a whopping five hundred and eighty five applications. These were whittled down to a final twenty, who were interviewed five future teammates and managers. Two out of the twenty were rejected based on their technical skills. Two were hired. Sixteen, though, were rejected because of personality clashes: “He just didn't feel right for the team”; “I couldn't imagine working with her”; “He wouldn't fit in around here”.

When people are rejected based on personality clashes, that usually means culture clashes. The candidate has an incompatible outlook on how teams and organizations should operate. Hiring them would be like putting a fox into the hen house.

Corporate Culture

What is corporate culture? A common definition is "the way we work around here", but that puts all the focus on what people do, rather than why they do it in the first place. Companies change their work practices all the time, but that doesn't necessarily mean they have changed their culture. Maybe a better definition for corporate culture is "the way we see things around here". Finding a job applicant who is a good fit, then, means finding somebody who looks at things the same way.

Culture Clashes

Cultures clashes happen when metaphors collide. A corporate culture always has a dominant metaphor, which drives the way people see things and react to them. A culture based on the production line metaphor would have a hard time taking on board a candidate who believed wholeheartedly in customer satisfaction. A company dominated by the model building metaphor would struggle to accommodate a candidate committed to the production line.

Lack of awareness of the dominant corporate metaphor can lead people to underestimate the impact of culture clashes. Too often we imagine that people will somehow adapt themselves and fit in. People can't just decide to fit in – they either fit in instinctively, or they fight against the culture and are forced out by the company, or they grit their teeth and do a job they hate, until they can't stand it any more and they leave.

A company that had struggled with a number of projects believed that hiring better people would make all the difference. Senior management announced that from now on they were only going to hire “the best”. By this they meant people with substantial experience in the latest technologies and practices. High salaries were dangled, lofty promises made, and several new faces appeared in the company.

The first few days were pretty exciting, but soon things started to go downhill. The new hires were told to do great work, but weren't allowed to change anything. Management could not accept the disruption of changing current processes and policies.

I know somebody who experienced the culture clash first-hand. Having gone through a recruitment process lasting several months, he resigned within weeks of starting the job. Senior executives were keen to keep him, and offered him almost any position within the company that took his fancy. After searching around for a week he said "There is no opportunity for me to make a difference here. The job titles varied, but the culture didn't." and so he left.

Managers had hoped that mere brain power would be enough. The new hires, though, were stifled: unable to work in the ways they knew to be right, and forced to work in ways they were sure were the cause of all the company's troubles. In most cases, the new hires left quickly. A few learned to comply: they became mercenaries; doing work they hated and seeing the high salaries as compensation for the frustration. “The best” had turned out to make no difference at all.

Forcing somebody to fit into a conflicting culture stifles the very things they could bring to the company. The simple message is, don't bring in people to shake things up unless you are willing and able to take the shaking.

Changing Cultures

Now, imagine you want to shift the current corporate culture, say from the production line metaphor to the model building metaphor. You problems are far greater than if you hired in a single person with a conflicting culture. If everybody in the company currently lives and breaths the production line metaphor, you have a culture clash with everybody.

There is no escaping that fact that changing cultures is hard. It means shifting everybody in the company to a new metaphor. Having said that, the steps are clearly defined, as are the many mistakes which can be made, and must be guarded against along the way. The next chapter lays out precisely how to go about changing your corporate culture, so that you take the whole organization with you.

Chapter 14: Changing the Corporate Culture

Change is Hard

Companies can change their cultures. I have seen it happen. Despite what some folks may claim, though, changing corporate culture doesn't come easily. It means shifting the company's dominant

metaphor, and that goes beyond merely changing behaviors; it means changing beliefs, and ultimately, values. For that the organization has to be ready, willing, and able to undergo major transformational change. This takes serious commitment, and more often than not involves a long and painful upheaval.

A lot of companies simply aren't ready, or willing, or able undergo the level of upheaval that cultural change entails. Many say they want to shift cultures, but when it came down to it, they want the rewards without the effort. They hope to sneak change in under the radar. At best, they gain some small and localized benefits, but it goes against the grain of the organization and the results are almost always disappointing. In the long run, they usually end up right back at square one.

I received a phone call from a senior executive at a well-known dot com. Let's call him Peter. He was flying into town, and wanted to take me to dinner. Over Mexican steak and red wine, Peter told me he was nervous. "Anthony, our biggest competitor has been taking all our business for the past two years. We are now playing catch-up rather than leading the pack. The more we try, the further behind we seem to get. What do you think they are doing that we are not?"

I knew the competitor well. I had spent time looking at how they worked, so I could tell Peter straight away. "They are using Lean Software Development", I began, then started to explain the approach in greater detail. Peter stopped me within two minutes: "Great. We want the same for our company. We want to go faster than everybody else, and we want to eliminate all waste. But we can't do it the way you described. We have to keep the good practices we already have in place."

A few days later, Peter put me in touch with Dave, a colleague of his. "We are going to do it!" Dave exclaimed, "But we've got to be practical. We can't afford to drop discipline. We aren't going 'official Lean'. Instead, we are going to be much more aggressive about project deadlines."

It struck me that Peter and Dave were scared of change. I was more scared that they were not going to change. After all, the competition was was beating the hell out of them in the marketplace. Peter and Dave's instinctive fear almost certainly emerged from their deeply ingrained production line metaphor. They believed strongly in a defined process with heavy management control and oversight, which they saw as inherently good, and letting go of this 'discipline' was seen as inviting chaos and danger into the company. Before Peter and Dave could lead a culture change, they would have to believe in it themselves, and this would mean accepting that change was going to be difficult, painful, and, yes, frightening.

I often explain to clients that a culture supported by an unwillingness to go through painful change is like a rubber band. You can stretch it, until it snaps under the strain, or you can give up, let go, and watch it twang all the way back to the point where it started.

Sneaking and tweaking simply isn't going to give the level of results companies are looking for. Shifting a corporate culture from one dominant metaphor to another is a huge transformational change. Like any major change, it won't happen by chance, and it won't come easily. It needs concerted effort. Where I have seen success, it invariably comes down to the willingness of the people in the organization – from senior executives down to teams on the ground - to make the cultural shift no matter how painful.

A consultancy company hired me to shift their culture from production line to customer satisfaction. The company was lucky: senior management had realized they had a problem, and so did most employees. They didn't blame the problems on lack of compliance with the rules they already had in place, they saw they needed to change the rules.

When I arrived, I found analysts sitting in a different part of the building from developers, and documents passing back and forth between them. Everybody seemed glum, and work seemed to progress very slowly.

One of the analysts complained to me that the programmers were either unwilling or unable to follow through on implementing specifications. I walked next door, and both programmers poured their hearts out with tales of woe, where the analysts just slid specifications under the door rather than talking with them. The analysts and the programmers seemed like nice folks, but they had become locked in battle. The analysts believed wholeheartedly in the Production Line metaphor, and the programmers believed wholeheartedly in the Model Building metaphor. There was simply no common ground for them to gel around. Instead, "birds of a feather flock together", so they had separated into two us-and-them groups.

The first shift I made was to get everybody sitting in the same room. Twice a day for the first couple of weeks, we all chatted about current projects and noted important things on flipcharts as we went along. The idea was to slowly shift people's thinking from "me" to "we"; to get everybody involved, keep everybody informed, and keep everybody contributing.

Over the next few months, we gradually nudged the focus away from production and towards tracking customers' changing needs. This involved modifying both behaviour and language. It meant people had to learn to stop asking what they should be doing, and start asking what customers needed most. Instead of looking inwards, people slowly and painfully learned to look outwards.

One of the analysts couldn't cope with these changes. He was unwilling or unable to let go of the production line metaphor, wherein his specifications told programmers exactly what to do. He saw increased collaboration as a demotion. So, by mutual agreement, he resigned and left the company. Most people, though, experienced early discomfort but eventually took to the changes well.

As people slowly modified their language and their behaviour, they did begin to alter their whole outlook. The whole company gradually shifted its culture from one based on a production line metaphor to one based on customer satisfaction.

As the corporate culture shifted, so did morale. A few months in, one team member said to me "For the first time in years, I woke up this morning excited about coming to work". There was a more positive and pleasant atmosphere around the place. Things started getting done. Productivity increased. The company grew.

Three Vital Steps

Changing a corporate culture, then, doesn't involve wishful thinking. In fact, it involves three very clear steps:

- **Getting ready for change**: preparing the organization so that the change initiative can be more than empty slogans
- **Making change happen**: pushing change through, no matter what it takes
- **Making the change stick**: ensuring the new culture seeps deeply and permanently into and throughout the company, so nobody slips back into old habits

All three steps are vital. I have seen a great many companies full of enthusiasm for change, charging ahead with the second step, forgetting the first and the third. This is always a mistake.

Forgetting the first step of preparing the company for change almost always guarantees that the new

culture will never get off the ground.

Forgetting the third step allows the new culture to fizzle out. It takes time for the new culture to become sufficiently deeply ingrained within the company to prevent the old culture from creeping back in.

Eight Classic Mistakes

John Kotter, of the Harvard Business School, is perhaps the worlds leading expert on change. Kotter has spent years working with companies undergoing major transformational change, and found eight classic mistakes that companies make, each of which can derail change initiatives.

Kotter's book, Leading Change, lays out the eight mistakes, and emphasizes the importance of addressing each of them one by one – and in the order listed below – in order for change initiatives to have a chance of succeeding. Giving into pressure and jumping over any of the eight, or moving on from one to the next too early prevents transformation efforts from taking hold.

The eight classic mistakes are:

- Allowing too much complacency
- Failing to create a sufficiently powerful guiding coalition
- Underestimating the power of vision
- Under-communicating the vision
- Permitting obstacles to block the new vision
- Failing to create short-term wins
- Declaring victory too soon
- Neglecting to cement changes firmly in the corporate culture

In my own work with clients, I have come across all of these eight mistakes, and seen first hand the impact they have. Over the past few years, I have come to believe very strongly that focusing companies on tackling Kotter's eight classic mistakes gives them an excellent road-map for shifting corporate culture from one metaphor to another.

Let's look at Kotter's eight classic mistakes, this time grouped against the three vital steps of effective cultural change.

Step 1: Getting Ready for Change

Remember, cultures do clash, and the dominant beliefs of the current culture will resist the beliefs of the culture you are aiming for. It takes major preparatory effort to reduce that resistance. It is only when an organization is *ready* to change its culture that the organization will be *able* change its culture. The first four mistakes are symptoms of an organization that is too frozen for cultural change to be possible. Such organizations need to be thawed out before change can be implemented.

Mistake 1: Allowing too much complacency

The biggest mistake is charging ahead without first creating a sense of urgency throughout the company. If the general mood is that there is no immediate need to shift the culture, or managers foster caution and a false sense of security, then there won't be enough drive to make change happen.

> Three years ago, a software company in the UK approached me to help "move the business into the modern world". They were losing customers rapidly, and revenue was sliding. After digging into the issues, I made a bunch of recommendations to move the company to a customer satisfaction culture, and the partners nodded enthusiastically and committed to making these changes happen.
>
> At first, though, nothing happened. I pushed a little. One partner sent out a memo asking for volunteers interested in participating. A couple of people showed interest, but soon dropped out. I pushed some more. Still nothing happened. Eventually, one of the partners took me aside and explained: "We have faced hurdles many times over the past twenty years, and somehow we always got over them. Things will most likely settle down and be back to normal in a few months". With no sense of urgency, the business only had half-hearted commitment to making change happen.
>
> Two and a half years later, things were now critical: the company was close to going out of business. They took on new senior management, specifically authorized to do whatever it takes to rescue the business. That included taking on two of my team, to work full time with them implementing the changes I had recommended almost three years earlier.
>
> Things did turn around, but it was a close call. With a greater sense of urgency, and far less

complacency, the company would have saved themselves a lot of headaches, a lot of time, and would be in a far stronger financial position than they are today.

Urgency, by the way, is not anxiety; it is a compelling will to go beyond empty talk and take actual action to make change happen.

An ex-colleague, Rodrigo, worked at a company whose senior executives made great fanfare about an imminent shift of the business to be participatory rather than dictatorial. From now on, the company sought innovation.

This sounded great, and enthusiasm and employee morale were at an all time high throughout the organization. Until, that is, employees saw the execution of the vision: a suggestions box, into which they could drop their innovative ideas. There was a prize for the best idea of the month. Rodrigo won a couple of times. None of the prize winning ideas were ever actually implemented. Rodrigo soon stopped innovating, and so did everybody else. Why bother?

Eventually the suggestions box was removed quietly, and that was the end of the grand plans for the innovation throughout business.

Mistake 2: Failing to create a sufficiently powerful guiding coalition

Major cultural change is impossible without active support from multiple people at the very top of the organization. Single individuals, powerless committees, and delegated task forces are going to be ineffective. A powerful and credible senior leadership team is vital to overcoming the major sources of inertia that undermine effective change initiatives.

That leadership team cannot be dominated by managers who believe wholeheartedly in command and control. If it is, they are going to have a very hard time leading cultural change. Command-and-control managers tend to see calls for change are as dissent. They don't innovate, they administer. They have a habit of enforcing the status quo. When they do seek change, they try to implement it dictatorially.

Simply commanding people to change isn't going to work. People might follow orders, but they won't necessarily believe in them. They will be acting the part rather than living it. More than ever, it is the willful cooperation of the people on the ground that makes change initiatives succeed. For that to

happen people don’t need more management, they need more leadership.

Leadership isn't easy. Warren Bennis has likened leadership to herding cats: people cannot be pushed; they must be pulled gently. Leaders look to innovate rather than administer and control. They are always looking for ways to improve things for the long haul. They challenge the status quo, and see calls for change as valuable feedback worth exploring. Above all, leaders are out to inspire, and they lead by example rather than by command.

> One of my former employers wanted to shift from production-line command-and-control management to a more collaborative style. The managers deemed themselves too busy to get involved, so they assigned two junior employees to lead the change. These employees were toothless: they travelled around the company making presentations, and encouraging people to change, but were generally given the brush off. When the two employees made their final reports to management, they were given the corporate equivalent of a pat on the back for effort: dinner with their spouses on company expenses. The change effort fizzled out, the the two employees moved on to other tasks.

Mistake 3: Underestimating the power of vision

Cultural change needs to be driven by a feasible, well articulated, and inspiring vision of the future. This is essential to providing clarity, guiding effective decision making, and getting widespread commitment to common direction and goals. Far too often, companies lose the vision, and end up immersed in detailed and complex management plans, with the change process swamped in trivial debates and stifling bureaucracy.

> At the European head office of a large American company there was an excited buzz in the air. A top executive was flying over from the US to reveal the company's vision for the coming year.
>
> I stood to the side of the room and watched as the speaker began his presentation to an eager audience of several hundred. It went something like this: “I know it may not seem that we have a long term plan, but we do. We know exactly where this company needs to go. We know all of the decisions that have to be made, we have made made many of those decisions already, and we have planned ahead for the decisions yet to come.” Then, after a short pause: “Are there any questions?”

The first question was obvious: “Can you tell us what those decisions are?” To which the executive replied: “Each decision will be revealed to you when the time is right. As decisions are made they will be passed to your managers. Your job will be to ensure they are carried out. And now, ladies and gentleman, pizza and soft drinks are available at the back of the room.”

How could anybody be inspired by that?

As I ate pizza and drank coke I overhead exactly how people felt about it: “That wasn't a vision statement, that was a bullshit statement”, “What he means is 'We have plans, but they are too secret to share with you nobodies'”, “I feel like going home!”

Mistake 4: Under-communicating the vision

A vision can't be dictated. You can't force people to buy into it, while ignoring their opinions and brushing all their questions aside. As the own proverb goes “A man convinced against his will is of the same opinion still.”

To capture hearts and minds, continual two-way communication is vital. Not just in a flow of compelling words, but in a continual flow of inspiring deeds. Senior managers have to lead change by example. If they say one thing and do another the process will lose all credibility.

Many years ago, not long out of university, I joined a company that declared to all customers “we are staking our future on the endless pursuit of quality”. A new Quality Team was formed, to go on road-shows and spread the message. Employees were given stickers and badges announcing that “Quality is Job One”. Quality was everything. At least in words. There was no actual follow through. It was all an empty marketing campaign.

A couple of months down the line, the marketing department sent a memo around saying that hundreds of umbrellas were left over, each announcing “Quality First”. These umbrellas had been free gifts to customers, but few had been taken, and to reduce the over supply each employee was required to buy one. What an insult!

A few days later, a second memo was sent around stating that since few employees had been willing to buy an umbrella, it was evident that employees didn't believe in the quality

campaign. Too right!

Step 2: Making Change Happen

Once an organization has been thawed out, it is ready for change, but is it willing and able? The next three mistakes undermine the effective implementation of change.

Mistake 5: Permitting obstacles to block the new vision

This is a tough one, since cultural change affects everyone, and resistance can come from anywhere. There may be an organizational structure that continually undermines change, or a performance appraisal system that punishes people for changing the way they work, or supervisors who enforce old ways of behaving, or insufficient training to help break out of ingrained habits. If there are things in the organization that block employees trying to follow the new vision, they will feel powerless to make change happen. It requires immense commitment and action by upper management to permanently remove all such obstacles, as the first move in shifting cultures. This means redefining corporate structure, creating a completely new performance measurement and reward system, training managers and staff, and generally doing whatever it takes to reshape the company for the new culture.

Two members of my team were working at a company on a telecommunications project. This was an important project, since telecommunications was a new business area for the company. End-user needs were unclear, and so requirements and design would be allowed to emerge over time.

Word from the top was that this project would herald the start of the company's shift to Agile development.

After a few days on the project, my team of two was asked by company managers for a status report. They gave an honest account of work done so far, and explained which things were still unclear. Alas, they were rebuked for being “wishy-washy” with their project, and were forced to go back and create a highly detailed project plan.

Mistake 6: Failing to create short-term wins

Transforming an organizational culture takes time. Without early and continual results it can run out of steam. Rather than chasing the big dream and hoping things will work out in the long haul, management must actively foster short term successes along the way.

Let the results speak for themselves. When people see real and visible results, and see everybody celebrating them, they learn that the effort is worthwhile, the process gains credibility, and it keeps the momentum going.

It is important, though, that the results and the celebrations are genuine. Artificial results and sham celebrations can demoralize those who find out about it, and give a false sense of security to those who don't.

Several years ago, I shared an office with three people who had been struggling for close to two years to develop software performing complex financial analysis. The head of the department was under immense pressure to deliver. He announced that the company was shifting to monthly releases, and the first release would be on Friday. The three person team protested that they had nothing useful to release. “No problem,” replied the head of department, “It will only be a demo. I will get the raw numbers a day ahead, and you can write a program that fakes the calculation.” The team was horrified, but bowed to the pressure.

The day of the demo arrived – and the head of department brought with him the head of the company. He gave a little speech about this being a major milestone. The team was instructed to demonstrate the software, which not surprisingly showed the results that they had pre-programmed earlier that day. The head of department beamed: “You will see even more progress at the end of next month!”

The head of the company was overjoyed, having waited for so long to see any results at all. “I have a surprise,” she announced, and in came a massive celebratory cake, and two bottles of champagne. She patted the team members on the shoulder, and shook their hands. She left the room very happy indeed.

The mood among the team of three sank to an all time low. Not only had they celebrated completely faked results, they were now committed to doing the same next month. Within a

week, one of the team was searching for a new job, and a few weeks later he was gone. Another asked to move to another project. Only one remained on the project, which a couple of month later was scrapped after some very heated meetings between the head of department and upper management.

Mistake 7: Declaring victory too soon

If early success leads to premature completion of the change process, the results will be fragile. Declaring victory too soon prevents the new corporate culture from becoming sufficiently deeply ingrained. It can take years of continual reinforcement of new behaviors and attitudes, until they have seeped deep enough into the company for them to become nothing less than a complete cultural change.

I have made this mistake in my own work far too many times. For years, I was often dismayed to see changes unravel within months of me leaving client sites. The client's generally seemed happy with my work, but seemed unable to keep the momentum going once I had left.

It took me a long time to realize that we were declaring victory too soon. We were skipping the vital step of ensuring that the changes were ingrained deeply into the corporate culture. Nowadays, I pay much more attention to this, so that clients can become self sufficient.

At first, the new beliefs will only be in people's heads. They will be acting the new culture rather than living it; doing what they have been told is right rather than what they know and feel is right. A new culture hasn't truly sunk in until those new beliefs shift from being mental to becoming emotional. That is, the culture's underlying metaphor has to move from people's brains to people's guts. It is only then that beliefs become values, and only then that people will instinctively support, uphold, and indeed thrive in the new culture.

Without sufficient time, and without continual reinforcement, this shift from mental to emotional cannot happen. New mental beliefs will eventually give way to old emotional ones, and the old culture, and all its associated beliefs and practices, will come creeping back in.

The classic example of this mistake is the Chrysler Comprehensive Compensation (C3) project, where Kent Beck led the development a new payroll system at Daimler Chrysler. The project was a great success. It became the poster child for Kent's new eXtreme Programming (XP) methodology, leading to a breathtakingly rapid up-take of XP across a wide range of projects and organizations.

However, within months of Kent leaving, the C3 project was scrapped. Daimler Chrysler abandoned XP, and slipped back to their old ways of working.

Kent had left too early, and the project and the company had become too dependent on his drive and his charisma. To become self sufficient, they needed to allow the cultural change to sink deep into the organization before letting Kent go. As it was, the cultural rubber band twanged all the way back to where it had started from.

Step 3: Making the Change Stick

Addressing all the previous seven mistakes will help convert the current generation of leaders and staff to the new culture. There seems little point in undergoing a long and painful cultural transformation, though, if it doesn't survive to the next generation.

Mistake 8: Neglecting to cement changes firmly in the corporate culture

Once an organization has successfully transformed its culture, it needs to make sure that the culture will survive from one generation to the next. At any time, new leaders and new employees may come into the company, infused with cultures of their own. We need them to absorb our culture, rather than us to absorb theirs.

Careless hiring can undermine the new corporate culture. When hiring new people it is vital to keep the new culture in mind. Poor hiring decisions, particularly at the senior management level, can lead to strong personalities unable or unwilling to fit in with "the way we see and do things around here".

Sometimes people will simply not fit in, and it is almost certainly better to let them go, than to allow them to undermine the corporate culture. Once we are sure the right people have been hired, we need the current generation to live and breath and continually promote the new culture, and ensure it passes

on from this generation to the next.

Section 6

A Multi-Metaphor Exploration of RUP

Hopefully you now have a feel for several metaphors of software development. This should help you evaluate methodologies from a wider range of perspectives than a single metaphor alone. To give you a example of this in practice, this section presents an overview of a popular software development methodology: the Rational Unified Process (RUP). Plenty of quotes are included from published books describing the RUP.

As you read about RUP, reflect on your own reactions to it, and see if you can figure out where those feelings come from. Perhaps more importantly, try to work out which metaphors the creators of RUP subscribe to. If you find that parts of the RUP in conflict with one another, ask yourself why that is? Ask whether different authors of the RUP were subscribing to different metaphors. Does RUP blend them well, or it is a no-win compromise of conflicting beliefs?

Appendix A : Other Metaphors

Introduction

The Mathematics Metaphor

This metaphor has historically been popular in academia, although this appears to be on the decline. It has not proven popular in industry.

Perhaps the strongest proponent of the Mathematics metaphor was the late Edsger W. Dijkstra. Many of Dijkstra's writings are now available on his web-page at the University of Texas. They are well worth reading. Here I will quote from those papers, and use the names that Dijkstra used for them himself (e.g. EWD268).

In the 1950s, Dijkstra wrote software for computer hardware that had not yet been built. He only had specifications for the hardware. Therefore he could not actually test that his software worked. As a result, Dijkstra turned to mathematics, pencil, and paper, to prove with that his software was correct.

Dijkstra was delighted with the outcome, finding mathematical proof vastly superior to testing. This led to his now-famous remark in 1969 that “Program testing can be used to show the presence of bugs, but never to show their absence! Therefore, program correctness should be proved on account of the program text” (see EWD268)

Dijkstra then dedicated the remainder of his life to developing and evangelizing software development as a mathematical activity. His call in the early 1970s was for mathematical proof that programs were correct. Later on, this was supplemented with his assertion that the proof should actually precede the program, forming its specification.

> Dijkstra actually argued against the use to metaphors to describe or think about software development. Computers represent such a “radical novelty”, he argued, that “metaphors and analogies are too shallow” to be of value (see EWD1036). Instead, we must face up to the reality that software development is not about writing programs that run on computers. Oh no, that is merely a side effect. “It is not unusual – although a mistake – to consider the programmer’s task to be the production of programs ... It is an intrinsic duty of everyone who professes to compose algorithms to supply a proof” (see EWD316) In other words, the main activity of software development is to create mathematical proofs that the program

you write would run correctly if you later chose to run them on computers (see again EWD1036).

This, of course, is back to Dijkstra's own metaphor that programming is a mathematical activity. Of course, Dijkstra didn't accept that it is a metaphor. To him, it was a fact. This is not uncommon. I have seen plenty of people hold different metaphors as deeply as did Dijkstra, and each tended to interpret their metaphor at literal fact. Only other people need metaphors, because they are unable or unwilling to see the absolute truth that you see so clearly. Naturally, they think the same about you.

Dijkstra was particularly ashamed of the state of the computer industry. Throughout the 1970s and 1980s, he argued repeatedly that creativity, innovation, and shrewdness had no place here. “It has greatly saddened me to see that industry on the whole persist in a behaviour as if common sense suffices where I know that the techniques of scientific thought are required.” (see EWD917). The sloppy computer industry needed to be saved from itself, and computer science steeped in mathematics was the only possible saviour. Dijkstra made it his mission to help the computer industry transform itself from craft to science.

To Dijkstra’s increasing dismay, most people in industry showed little interest in his evangelical message. They were, instead, buying “snake oil”. In the late 1970s, he expressed particular anger at the emerging use of diagrams and graphical notations which he saw as a “crutch” for “amateur thinkers”. They are, he believed “for the uneducated and by the uneducated” (see EWD696). “Society needs competence and asks for quackery” he lamented (see EWD966).

In mid 1980s he noted that industry simply wasn't interested in improving: “They only have the choice between incompetence and dishonesty. It is terrible. But, please, don’t blame us when the computer industry collapses.” (see EWD917)

If only industry would listen, since “An academically educated computing scientist should be able to program at least an order of magnitude better than the average programmer or hacker without formal training” (see EWD1157). Alas, “the systematic disqualification of competence … is the managers’ own invention, for the sad consequences of which they should bear the full blame.” (see EWD966)

Managers in particular, are to blame for their evident mission to reduce dependency on skilled employees with a belief that “second-rate intellects are good enough for the industrial enterprise” EWD966 The “organizational love of mediocrity” is particularly evident in management since “the best professionals are the least inclined to switch to a managerial position” EWD966

By the late 1980s, as well as blaming managers, Dijkstra was now also blaming programmers: “Real programmers don’t reason about their programs, for reasoning isn’t macho. They rather get their substitute for intellectual satisfaction from not quite understanding what they are doing in their daring irresponsibility and from the subsequent excitement of chasing the bugs they should not have introduced in the first place.” (see EWD1012)

By the early 1990s Dijkstra appears to have almost given up hope of converting people in industry, due to their inferiority: “The mediocre mistrust the exceptional man because they don’t understand him, and they fear him because he can do things beyond their comprehension” (see EWD1165)

He wanted academia to fight back. “For 40 years, the computer industry has completely ignored the findings of computing science ... There is a marked discrepancy between what society asks for and what society needs ... It is the task of the first-class university to tell industry what it does not want to hear … to ensure that the sting of the academic gadfly really hurts” (see EWD1165)

He had begun, though, to feel that it was a lost cause: “The battles are between the managers/beancounters on the one hand, and the scientists/technologists on the other … Scientists have a tendency to lose them … because their primary interest is their science … in contrast to the manager, who considers this battle his main challenge.” (see EWD1165)

By the mid 1990s he reflected on the “cruel twist of history” where 20th century society had widely shunned mathematics. “Yet we cannot blame the universities”. The blame lay with industry for pressuring universities to “not indulge … in scientific education, but to confine itself to vocational training” (see EWD1209). Towards the end of his life, Dijkstra worried for the future of university education, and pleaded “as long as computing science is not allowed to save the computer industry, we had better see to it that the computer industry does not kill computer science” (see EWD1284)

Requiescat In Pace.