Software Project Cultures

Anthony Lauder

2008

Preface

Why did I write this book?

I started writing this book nearly ten years ago, because colleagues, clients, and friends had asked me to. What these people all had in common was a feeling of being overwhelmed by the huge variety of software development methodologies, each evangelizing its author's own “rules” for running projects and developing software.

These people told me they were struggling to work out what made sense in practice and what just looked good on paper. They told me they just couldn't get the big picture; they were swamped in details. So I started writing a book aimed at people just like them. It surveyed a wide range of software development methodologies, and explored how they related to one another and how they differed.

After several months bashing away at the keyboard, I released several chapters for review, and was aghast when some of the very same people who had prompted me to write the book replied: “Interesting. But what is the practical value?”

My ego was somewhat humbled, and for a while I stopped writing. Yet the question “What is the practical value?” never went away. It was always nagging in the back of my mind.

Over the next few years, as I went about my consultancy business, I began to notice other people asking related things. Questions such as:

- Why does my boss force a methodology on us that is so clearly idiotic?
- Why won't the programmers do as they are told?
- Why can't I get methodology XYZ to take hold in my company?”

It slowly dawned on me that people weren't looking to be told *how* they *should* work, but rather wanted to understand *why* different people *do* work in one way or another.

They weren't asking me so much about methodologies as they were asking about how methodologies survive or fail in specific software development cultures. Not only that, people then started asking me:

- If that is our culture, then how do go about changing it?

How is this book different?

This distinction between methodologies and culture seemed important. Nobody was writing about it, but people were asking about it. There was clearly a gap in the market, and it sprang me back into writing again. The result is this book.

This book shows how awareness of and sensitivity to software development cultures can make the difference between success and failure on real-world software projects. Rather than spouting mere theory, the book digs deep into many real life example, to uncover what software development cultures are, where they come from, and the impact they have. Three dominant cultures are identified and explored in detail, along with a fourth culture, which is now mostly dormant and is relegated to the appendices.

For each of the three dominant cultures, we look into its history, it underlying cultural assumptions, and the software development methodologies that are reflected by that culture.

Armed with this deepened cultural awareness, the book then turns to the very tricky question of how to highly successful organizations have successfully changed their culture. As we shall see, for cultural changes to take hold it takes far more than imposing a new methodology on a team. Instead, a software development culture must be slowly nudged up a cultural ladder, upgrading that culture one rung at a time.

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

Chapter 1 : Introduction

Culture Matters

If you only take one message away from this book, then, let it be this: culture matters. It matters a lot.

A software development culture is not the same thing as a software development methodology. You can think of a methodology simply as a set of rules to be followed, whereas a culture is a set of deeply held beliefs around which a group of people have gelled. Methodologies might tell people how to behave, but cultures determine how people feel compelled to behave.

Cultures influence people's emotional reactions to the rules imposed on them. If a methodology and a culture don't match, people will instinctively feel anxious, and the methodology will meet resistance. If a manager is committed to one culture and his or her team has gelled around another, they will both feel anxious, and there will be a painful culture clash. The relationship then becomes antagonistic, and progress is hindered.

Of course, if you do have a bit of power over someone - like a manger over a team for example - you can certainly force them to do what you want. That's fine if all you want is obedience. It's certainly the way prison guards get chain-gangs to smash rocks all day. Unfortunately, I have seen time and again that This approach simply doesn't work very well for businesses involved in developing software. Here is why: Although bullying and threats can make people obedient, it cannot get their willful cooperation. You can shout, scream, and punish all you want, but without willful cooperation people will never put their heart and soul into what they do. The results will be mediocre at best.

If you want great results, you have to take culture into account.

Follow Your Emotions

We will jump right in and start with a multiple-choice quiz, that will help explore your own instinctive reactions to several different aspects of software development. YouR responses should highlight just how powerful emotions are and how strong their influence can be in pulling you toward one culture over another.

When answering these ten questions, rely mainly on your gut instincts. Note which answers make you feel anxious, and which are comforting. Sometimes more than one answer will ring true, 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.

After you have finished the quiz, we will review your responses. The results should hit home the point that your emotional reactions make you more sympathetic towards some software development cultures, and rather antagonistic towards others.

Ten Questions

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 Technical Team: Management is essentially administrative, and when it comes down to it contributes little to the actual creation of great software, which requires above all a talented, experienced, and very technical software development team.

C: Customers: Without customers there would be no software. Only customers know what they really need and value, so customers must remain firmly in the driving seat to ensure we remain focused on continually delivering the software that matters to them most and that solves their most urgent business needs.

Q2: How can we best get the software requirements?

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 usually pretty unclear at first, but details do become clearer over time, at least on the most important requirements, and these help us increasingly stabilize the design of the architecture upon which we build the rest of the system.

C: Expect them to Change: We cannot expect requirements to ever stabilize. Previously important requirements lose their importance over time, and unexpected events lead to the emergence of completely new and unanticipated customer needs. We should remain focused only on those requirements that the customer currently values most highly and expect those priorities to change continually.

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

A: Textual Documents: Write documents, primarily in text, but with supporting charts and diagrams where helpful, 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 Notations: Create diagrams, using a relatively formal graphical notation, to express requirements and designs more precisely and concisely than is possible with ambiguous text, perhaps with support from a software tool that "understands" those diagrams and supports your creation of them.

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

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

A: Transformation: Requirements should be translated into a detailed design document, 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. We can then reflect those abstractions in the design and implementation of a robust software architecture upon which the rest of the software can then be built with confidence.

C: Delivering Software: Documents, models, and other interim products are only useful if they help get software into customer hands. Therefore, put the main focus on delivering working software quickly and often, no matter how basic at first, so that customers can benefit from the value of that software as soon as possible.

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

A: Plan and control: 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: Lead: Put the right people on the project, help them gel as a team, give them guidance when they ask for it, but get out of the way while they deliver results quickly, without your potentially stifling interference.

C: Support: Encourage the team to bring to your attention any obstacles they find in their path that hinder their progress or threaten their work pace, then support the team by using all your powers to remove those obstacles and thus increase the team's ability to succeed.

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 work as efficiently and effectively as possible and thereby 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 partial slice in 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 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 results confuse 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. Instead, focus primarily on modelling the core of the business

domain being supported by the software. Using software tools to capture those models provides 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 delivering immediately high business value. Low-priority requirements are not worth up-front investment, and if and when their business value did increase changing circumstances mean they would likely have to be redone anyway.

Q10: What indicates that a software project finished?

A: Tasks Completed: When all planned tasks have been completed correctly, the project has finished.

B: Stability: When the software design is sufficiently stable and trustworthy, the software can be released into customer hands.

C: Customer Value: When the customer decides that the next version of the software won't deliver enough extra business value to warrant the cost, we stop working on the project.

How did you Score?

Some of the answers above may have resonated for you, and others may have seemed disturbing, ludicrous, or even complete mumbo-jumbo. It is quite possible that your preferred answers were a random scattering of As, Bs, and Cs, but more likely they followed a pattern. Most people do seem to have gut instincts that guide them more towards one set of answers over the others.

Those instincts are important, because they shape the way we feel about and how we are likely to react to all kinds of other software development issues. They will draw us towards particular software methodologies based not on analysis of each and every rule, but on how right they feel overall. 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. Therefore, your answers to the quiz don't just help uncover some of your own emotional tendencies, they also help work out the type of group culture in which you would likely be most comfortable.

If you find that you did choose mostly As, mostly Bs, or mostly Cs, then you may well find that the following rings true for you too:

Mostly As

- 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

- If you are actively involved in software development, you may well be using or at least drawn to a Waterfall methodology, such as Gane and Sarsen, or Systems Analysis and Design Method (SSADM). More on this in chapter XXX.
- You will probably fit well into an organization with a dominant Manufacturing Culture. There is a brief overview of this culture, and the other two dominant cultures, later in this chapter. It is probably worth the wait, but if you are impatient you may wish to skip forward a few pages for a quick peek. If you are particularly anxious to find out all the gory details you can even leap ahead to chapter XXX.

Mostly Bs

- 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
- If you are actively involved in software development, you may well be using or at least drawn to an Object-Oriented methodology, such as XXXX or XXXX. More on this in chapter XXX.
- You will probably fit well into an organization with a dominant Design Culture. Chapter XXX which explores the Design Culture in depth.

Mostly Cs

- 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 keeping up with the customer's ever-changing needs.
- If you are actively involved in software development, you may well be using or are at least drawn to an Agile methodology such as Scrum or Crystal. More on this in chapter XXX.
- You will probably fit well into an organization with a dominant Service Culture. All the juicy details of a Service Culture are revealed in Chapter XXX.

Real-Life Example

Hopefully, this quiz has highlighted for you the personal emotional pull of different cultures. This may help you see why there is little use imposing a new and promising methodology on a person or a group if that methodology conflicts with their own dominant culture. An imposed and conflicting methodology will cause them to feel great anxiety, and they have no confidence in it. They will resist that methodology instinctively, and will have doubts about your judgements for imposing it.

The problem is that imposition can only dictate changes in behavior. It cannot change a person's or a group's dominant cultural mindset nor the instinctive reactions that stem from that mindset. The results, then, will likely be dismal. Recapping the main message of this book, then, if you want great results, you have to take culture into account.

All of this could, of course, just be an intellectual curiosity. So, let me reinforce the message though a real-life example, where a well-known company had neglected its own software development culture, and suffered tremendous loss of market share as a result. Hopefully, this example will illustrate very clearly the major impact that software development cultures do have in the real world, and convey why they should matter to you too.

Competitive Times

A while back, I worked for a couple of months with a well known “dot com”. To protect privacy, we will call the company Self-Serve.

Self-Serve has revolutionized their particular industry, moving it from being human-intensive to being centered on Internet-based self-service. Self-Serve has done very well out of this, with an annual turnover of several billion dollars. Their market domination, however, has started to slip in a rather serious way. Over the past two to three years, several competitors have started to race ahead. They are grabbing market share at a frightening pace. Self-Serve's executives told me they are seriously worried.

Self-Serve hired me, and I spent a couple of months investigating what was going on.

I found that Self-Serve had a strong management team, great technology, super smart people, and a budget that was breathtakingly high. On the face of it, they seemed to be doing everything right. What they seriously lacked, though, was an ability to react and adapt the business quickly enough to beat off the competition. Sure, they felt they knew what needed doing, in terms of business processes that had

to be changed, and software systems that needed developing, but making it all actually happen always seemed to take an unbearably long time. It was like swimming in molasses.

A Manufacturing Culture

Something was holding Self-Serve back. Something was preventing the company from being able to adapt their business processes and their software systems at a rate matching that of their competitors. Every day that passed, Self-Serve was falling further behind.

They tried motivational speeches; they tried threats; they tried throwing money at the problem; yet none of it proved particularly effective. So, I asked around. I spoke with managers, I spoke with team leaders, and I spoke with people on the ground. What I discovered from talking with all these people was that Self-Serve had a deeply entrenched Manufacturing Culture, and it was holding the business back.

A Manufacturing Culture assumes that a project can and should be run like a production line in a factory. Product specifications should be pinned down in an up-front plan. Workers should be stationed in sequence along the production line, each following well-defined tasks against a time-schedule laid out in the plan. Each worker on the production line receives the output of the worker before them, completes their own tasks as described in the plan, and passes their results along to the next worker on the production line. Completed products roll off the end of the production line on budget, on schedule, and matching the specification. Managers ensure that workers adhere closely to the plan, and punish those who do not, since deviating from the plan threatens project success.

The assumptions that infuse a Manufacturing Culture sound reasonable, and therein lies the problem. They were indeed reasonable during the slow-moving business world of the 1960s and the 1970s, when the Manufacturing Culture was at its peak of popularity in the software industry. Way back then, computers were mostly aimed at technicians doing “number crunching” in the back office. Business processes rarely changed. Software system requirements were stable. Besides, as far as most people in an organization were concerned, in those days software didn’t matter all that much.

How times have changed. These days, software has become so vital to the efficient execution of any organization’s major business processes that it would be unthinkable to separate development of the two. In today's fast-paced business world, if software systems can’t keep up with the rate of business process change, then the efficient execution of those business processes is undermined, and software

begins to hold the business back. Software becomes a drain on the business: a business liability rather than a business asset. In these competitive times, no business can afford that.

Astonishingly, though, many companies, including Self-Serve, are still developing software as if business processes hardly mattered: using approaches to software development that were designed for the issues businesses faced way back in the 1960s and 1970s.

The rules that applied in those slower-paced, more-stable times simply don't apply any more. Nowadays, rather than contributing to success, they stifle it. This is a travesty. Imagine if any other part of your business worked in ways unchanged for forty years or more. You wouldn't tolerate it. Nor should you have to tolerate it for software development.

A Design Culture

Rather than fretting further over what Self-Serve was doing wrong, I resolved to find out what Self-Serve's most successful competitors were doing right.

It is surprising how much you can learn about a company's internal workings with a little digging around, and I soon discovered that a number of Self-Serve's competitors had turned their backs on a Manufacturing Culture and had cultivated instead a Design Culture.

A Design Culture embraces a very different set of assumptions about software development, in particular the assumption that future business process changes are undeniable. Software which does not anticipate such change is fragile, and soon fails to satisfy the business. Flexible software, on the other hand, is designed with such change in mind. Flexible software has a solid and stable architectural-base, with built-in placeholders that anticipate changes in the business processes being supported. This anticipation and preparation ensures that the software can be adapted quickly and therefore remain highly valuable to the business. Great architects create such architectures by making scale models to investigate and clarify uncertainties, to address tricky problems, to experiment with promising solutions, to anticipate and prepare for future changes, and overall to allow the design to settle down. Once the architectural design is stabilized, the rest of the product can be built on it with confidence.

By cultivating a Design Culture, several of Self-Serve's competitors were able to anticipate change and reflect it rapidly in their software, thereby gaining them a considerable competitive advantage on Self-Serve, which they exploited enthusiastically.

Still, this didn't fully explain why one of Self-Serve's competitors in particular was leaping far ahead of all the others and leaving them in its wake. This one company had to be doing something very different, and I resolved to find out what it was.

A Service Culture

It took a while, but I uncovered precisely how that one organization was leading the pack. It wasn't about the people, the technology, or the amount of money they threw at projects. If anything, they had fewer people, more humdrum technology, and a rather modest budget.

The distinguishing factor turned out to be that they had abandoned a Manufacturing Culture and a Design Culture, and were now totally united around a Service Culture.

As we have already seen, a Manufacturing Culture assumes that requirements can be pinned down up front, and it resists changes in those requirement. A Design Culture, on the other hand, anticipates and prepares for future business processes changes. This is a great improvement, but what happens when you face business process changes that you hadn't anticipated?

Making excuses that some important requirements change doesn't fit into the design isn't going to help the customer. It is going to hinder them, therefore a Service Culture gets us to shift our focus outwards and be driven by the changing needs of our customers, rather looking inwards for stability in the internal structure of the software.

A Service Culture has very different assumptions than either a Manufacturing Culture or a Design Culture. A Service Culture assumes that software must remain valuable in the eyes of the customer as the customer's needs continually change, even in unanticipated ways. Software, then, must remain soft, and its design needs to adapted continually, in response to such unanticipated changes. Customers should take a central role in identifying their shifting priorities, and determining which software features should be developed, and the order in which they are delivered. Delivering fast and often allows customers to use product features while they still have high business value. It also allows them to change priorities for which features are delivered next, and halt further development when cost outstrips potential value returned.

Why does this make a difference?

The Manufacturing and Design Cultures focus on *our* business success. A Service Culture, on the other

hand, focuses on supporting the *customer's* continual success. It recognizes that our customers are in continual competition in their own markets. To survive and thrive they have to respond to opportunities whenever they can, and this means adapting their own business processes in whatever way is necessary. By shifting our focus onto continually supporting those changing business processes, we become an asset to the customer rather than an expense.

This Service Culture, then, was the key to the extraordinary growth of Self-Serve's most successful competitor. They didn't waste money, time, and effort in guessing what their customers wanted. They invested instead in building relationships based on genuine trusted partnership, working closely with their customers to track and support continually their real and changing business priorities. Customers' saw that the relationship was contributing to their own success, and this was reflected right back in the success of this outstanding organization. It worked like a charm for them, and it hopefully demonstrates perfectly the profound effect a culture can have on an organization's prosperity.

Summary

To recap: you can have better technology, a higher budget, and better people than any of your competitors, yet be held back dramatically by your culture. If your organization clings to a culture infused with assumptions from from the 1960s and 1970s, you are only going to get 1960s and 1970s results. Your competitors will seize the opportunity to march ahead, and eventually put you out of business.

If you are absolutely positive this doesn't apply to your business, then you are one of the lucky few. You are already in a different league from those companies hampered by relics from the past, and this book can only confirm what you already know.

At the same time, nobody in their right mind is willingly using forty-year-old practices. Most businesses believe wholeheartedly that they are already doing all they can to remain ahead of the game. Yet more often than not, without even being aware of it, they are saturated in cultural assumptions that draw them to outdated software development approaches, and that holds them back from the level of results to which they aspire.

Highly successful companies, then, have highly successful cultures. If you simply try to copy the methodology that your most successful competitors employ, you will fail unless your culture can absorb it. Success comes not from selecting and enforcing a methodology, but from shifting cultures to

absorb those methodologies wholeheartedly.

This book will help you recognize your own software development culture. It will reveal the assumptions that compel you almost irresistibly to approach software development in the way you do. You will see for yourself which cultures hold businesses back, and which propel them forwards. Perhaps most importantly, this book will reveal how some companies have successfully upgraded their cultures, transforming software from a stifling business liability, into a world-class business asset. How you respond to that is up to you.

Chapter 2 : The Rules of the Game

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 praise to which he was surely entitled. He left in disgust. Strangely enough, projects on which he had worked started to improve. 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!”

> NEED AN EXAMPLE HERE

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”.

Playing by Different Rules

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 us 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.

> NEED AN EXAMPLE

The issue here isn't that “we” are the smart ones and “they” are idiots. It is that different people have very different beliefs, deep down, about what counts as project success, and how to achieve it, and what counts as project failure, and how to avoid it.

> NEED AN EXAMPLE

When it comes right down to it, software development is a game. In games, we make choices based on the choices of others. The rules of the game are the things we accept as being true in order to help us make those choices. With software development, nobody is quite sure what the rules are. Well-

established games, like Monopoly, Chess, and Football, do have well-established rules. Software development doesn't. It is a relatively new game, and people are still trying to figure out how the game should be played. In the absence of well-established rules, then, people give it their best guess. The result is that different people are playing the software development game by different rules.

Where, then, do different people get their different rules from? The answer seems to vary based on whether somebody is a beginner, has intermediate experience, or is an expert.

Beginners : Prescriptive Rules

Beginners are in a difficult situation. They lack software development experience and are therefore faced with immense uncertainty when dealing with even routine situations. To overcome this uncertainty, and the anxiety that accompanies it, they tend to look to more experienced people to provide prescriptive rules they can follow to the letter.

Plenty of experienced people are more than happy to share their rules, and the particularly keen package them up, evangelize them for reuse, and call them a *methodology*.

> 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, published by Addison Wesley in 1998. Here are some randomly picked 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.

There are other ways of packaging methodologies, as we shall see, but the most prevalent are prescriptive rule books. This is presumably because beginners form the largest audience for such books, and anything overly philosophical will be too abstract for beginners to act on.

Certainly, when I first started out in software development I would find thick and authoritative rule books far more reassuring than thinner books that offered only general advice. When I was at university in the 1980s, we were given some definite rules for software development: Draw these specific diagram; fill out these forms; design your code using these techniques; hand your design to these people for checking; and so on. It was all so tidy, and very reassuring to be following such clear and well laid out steps to success.

It was rather unsettling, then, when I first started working in industry, and one of the senior programmers said “Nobody does that in real life”. He wasn't a particularly good mentor, since he didn't offer me any alternative rules, other than “we have deadlines to meet” and “your code had better work”. I felt rather lost at sea to be honest.

Methodologies aimed at beginners, then, prescribe authoritative rules that must be obeyed, with their author's implicit promise that "these rules worked well for me, and if you follow them as described they will work well for you too".

NEED AN EXAMPLE

There are, or course, many competing methodologies out there, each advocating different prescriptive rules, and each vying for domination. The really arrogant have attempted to circumvent this methodology-war with a marketing ploy: calling their own prescriptive rules *best practices*. Their aim is to reduce a beginner's anxiety about adopting those rules, and increase their anxiety about adopting alternatives, which are then, by implication, *worse practices* that set projects up for failure.

The more faint of heart know they are somewhat misleading others with claims of universal best practices, and add a sneaky disclaimer to their methodology books: “Modify according to your specific project needs”. That's about as helpful as the cooking instructions that came with some very expensive pasta I once 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 of this sort are essentially marketing material for their author's consultancy business, helping confused people deal with the sticky bits

Despite the claims, no methodology has yet been established as *the* set of rules for software development. I have seen some consultants wriggle away from this discomfort with the rather slippery advice to their clients that: “There is no single best methodology; you have to select the appropriate methodology for your organization on a project-by-project basis”

While this advice may comfort the consultant, it is hardly helpful to their clients. Expecting beginners to become experts in all the methodologies out there, and then be able to select the correct one and adjust to it on a project-by-project basis simply isn't realistic. Besides, how can somebody determine the best methodology for a specific project until they have first used a methodology to dig deep into the project to unravel its specific needs?

> I worked for almost a year with the Space and Defence division of Logica in London. Project teams were expected to demonstrate they had chosen the best methodology according to individual project needs. In practice, project teams 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, using whichever methodology they favored personally. They then justified their "choice" with a flurry of after-the-fact paperwork based on uncovered project needs that they could not possibly have known up-front.

Given the allure of absolute certainty, then, almost all of my beginner-level clients have at some point pined for, and ultimately selected, a single methodology whose prescriptive rules they can use with confidence at all times. Some selected an externally-created methodology, having bought a book or been on a course, whereas others relied on rules copied from, or handed down to them by, old-timers in their own organization.

> Occasionally, I do come across beginner-level teams that claim they aren't follow any rules when developing software. In practice, I find this inevitably means that although they aren't following a written set of rules, the team has learned to follow conventions that wrap up the rules they follow implicitly.
>
> For example, Richard Smith and Mel Gibb were beginner-level programmers in a software development team at EDP, in Milton Keynes in the UK. Richard and Mel told me "We don't need no stinkin' rules; we just get on with projects". I was suspicious, so I listened to the 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, since I am still tweaking the functionality"

- "Its our turn to come to you guys this week to integrate our latest changes"

Even though the team "just get on with projects" they had clearly have picked up (from more experienced people it turned out) plenty of conventions wrapping the rules-of-the-game they follow without question, including:

- The boss may well be nasty, but his orders must be obeyed
- Periodically freeze new work and focus all attention on cleaning up code
- Stabilize functionality before optimizing for performance
- Team members must meet weekly to synchronize work

Having committed to a particular set of prescriptive rules, beginners tend to embrace them as absolute legislation, and can experience strong emotional reactions to any suggestions or behaviors that conflict with those rules. Anybody who breaks the rules is seen as irresponsible, even reckless, is endangering the project, and must be punished. There is then a very real danger that the methodology becomes dogma, which is followed with unquestioned devotion, even to the point where it is actually detrimental to projects.

I worked briefly with a large software company which had grown quickly. The company taken on a large number of relatively inexperienced newcomers, and handed them a very strict and heavyweight methodology, to be followed without question. This gave the newcomers immediate certainty about the work they should do, but it seemed to me that often more time was spent following rules than designing and writing software. Project managers and even fellow team members were told to be 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 more experienced 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 others as impractical. Under the radar attempts to work more effectively were

treated as insubordination by managers, and as rogue behaviour by more compliant team mates.

Project staff who didn't believe in the imposed rules were left with a choice: either keep others in the organization 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 of the doubters 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 enforcing 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.

Beginners, then, all too easily fall into a trap, where faith in the immutable and dogmatic rules of their chosen methodology blinds them to current alternatives and future improvements. As I explain to many of my clients, if you aren't willing to consider alternatives or take up improvements, you can be sure your competitors will. Despite these concerns, beginners appear to have a greater fear of uncertainty than of mediocrity, and feel an irresistible urge to follow fixed rules.

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 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 the SEI to strive to Level 4, wherein detailed measurements are taken 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.

Yet even commitment to a single prescriptive methodology is no guarantee of certainty. Disputes frequently erupt over what specific rules mean in practice. What was intended in some ambiguous sentence? When two rules conflict, which overrides the other? Can we tweak or even drop some particular rule if it don't seem to fit current circumstances, and still claim to be using that methodology? All too often, I have found that these conversations descend into very heated arguments over very trivial matters.

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 become involved in disputes over whether or not UML sequence diagrams were better. He seemed rather worried they had taken a wrong path, and 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.

This need for certainty, while seemingly comforting, underlies the most significant problem that eventually confronts beginners. They expect the methodology to answer every single question they come across. When this doesn't work in practice, they become extremely anxious. They have put a great deal of faith in their chosen methodology because they trust that its author knows more than they

do. Finding holes in the prescriptive rules casts doubt on the methodology and its author, and shakes the beginner's faith in both.

> In the past I have tried to explain to beginners that it is fine to look to a methodology for guidance, even as a very detailed level, but they cannot possibly expect it to serve as their complete rule book. The software development game is simply much more complicated than Monopoly, and Chess, and Football. I would usually get a nod of the head, but I could tell that my explanation was rarely persuasive. Often, I would be asked “So, which methodology should we be using instead?” It is hard, then, for beginners to drop the comfort of absolute certainty.

Many methodologists are sensitive to such gaps, and as they are highlighted, they tend to be filled in the next version of rule book. The results is that the prescriptive rules become more intricate, and hence the methodology becomes more dogmatic over time. This certainly makes the methodology less subjective, which might help solve a few disputes, but the danger is that it can also make the methodology less flexible and eventually too heavyweight to follow. Besides, it is impossible to plug all the gaps, so the struggle for absolute prescription is endless.

> 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.

Intermediates : Heuristic Values

Beginners, then, tend to place their faith in other people's prescriptive rules. Faith, of course, is belief without evidence. Beginners don't need evidence as much as they need certainty, so they follow rules without asking whether or not those rules are actually working in practice. That might be fine for beginners, but it usually isn't good enough for more experienced people.

Intermediates need more than faith. They want proof. If some rule looks good on paper, 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 actually works in practice will either affirm or weaken that faith. Rules that continually prove ineffective will eventually weaken to the point of collapse and will be rejected. Rules that are reinforced by repeated affirming experiences, will eventually become proven and trusted.

NEED AN EXAMPLE

As a result of this continual trial and error, intermediates rely less on prescriptive rules, and more on proven rules-of-thumb. They build a personal and growing body of heuristic values that help them judge what works, what doesn't work, and in what 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 prescriptive rules (the procedures manual) to the letter, and of course progress grinds almost to a halt. Knowing when and when not to apply the prescriptive rules is essential to working effectively. This is where experience comes in to play.

Heuristic values are more personal than adopted rules, because they are our own creations, born from the sweat of our own labours. They become things we care about deeply. As well as driving our actions and reactions, we use them to determine if we have been successful or not, and also to measure the success of others.

NEED AN EXAMPLE

Just as beginners can have strong emotional reactions to methodologies that conflicts with their adopted rules, intermediates can experience strong emotional reactions to methodologies that clash with their own heuristic values. Whenever we come across something that confirms our heuristic values, we experience warm fuzzy feelings, and it cements our heuristic values even further. When we come across something that conflicts with our heuristic values, we experience an uncomfortable emotional reaction to it. What we have just seen or heard won't *feel* right, and it will make us anxious.

NEED AN EXAMPLE

It is, of course, possible for an intermediate to write up their own heuristic values as a methodology for others to reuse, however this is a risky undertaking since beginners will not have sufficient experience to treat the heuristics as anything other than rules to be followed blindly. The effectiveness of heuristic values relies less on absolute obedience, and more on judgement and conviction grounded in personal experience. The major difference being that whereas a prescriptive rule is something you merely *think* is right, a heuristic value is something you *know* is right, deep in your gut, because it has proven itself to you time and again. Heuristic values, then, reflect the hard earned lessons of in-the-trenches

experience that differentiate the intermediate from the beginner.

> NEED AN EXAMPLE

Experts : Living Metaphors

We have seen that beginners rely largely on faith in borrowed prescriptive rules, which they accept as facts without need for evidence. Intermediates have internalized rules that proved effective, forming personal heuristic values. These help beginners and intermediates deal with the routine aspects of projects, where issues can be dealt with in ways that have been previously experienced or anticipated. Still, this doesn't explain how people deal with gaps, when confronted with unanticipated issues as projects unfold.

This question of how people deal with gaps in prescriptive rules and heuristic values has interested me for many years. As I worked with many team on their projects, it became clear that beginners and intermediates frequently fill in gaps by calling on experts.

At first, I assumed that experts simply had more and richer heuristic values, and I resolved to uncover them so that others could benefit from them. Over the years, I sat down with many experts and tried to tease out and capture their extensive heuristics values. Alas, this proved surprisingly difficult.

> I found that many experts had great difficulty articulating the heuristic values I was pressing them for. Often, they seemed to get very confused in the process, and we ended up recording rambling heuristics such as: “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!

The difficulties experts had in articulating their heuristic values made me suspicious. I began to wonder if experts were actually dealing with uncertainty in a different way. I began to suspect that they weren't calling on a deep store of highly complex heuristics after all, and that something else was going on beneath the level of heuristics.

> Clearly, it wasn't that experts were being deliberately deceitful or withholding their heuristics on purpose, it was that they seemed to be inventing heuristics on the fly. The answer to this finally hit me when talking to a fairly senior manager in a software firm. This guy had no programming experience on which to base his heuristic values, yet was supremely confident in his ability to run software projects. As I talked with him, it became clear that his heuristics were indeed, as with other experts, being generated on the fly. The new insight here, was that he was doing this by appealing to what had already worked in his previous career. He was performing mental gymnastics to transfer all his experience over from one business area (management consultancy) to another business area (software development). In other words, he was continually elaborating a set of deeply held metaphors to provide the rules and the heuristics by which he ran projects.

This, I have come to believe, is the different between experts and others: Experts call on living metaphors, grounded in successes they have already experienced, and which they continually elaborate to help them turn an unfamiliar situation into a familiar one. Metaphors are used by experts as torch-lights to show them the way.

> A metaphor allows us to approach, explore, reflect upon, and discuss a new area in terms of one we already understand.
>
> My brother, a software developer, was asked by our grandfather what his work involved. He tried explaining it literally, in technical terms, but it because clear that he was simply causing confusion. So, he turned to metaphor: “Programming is solving crossword puzzles all day long”.
>
> It was as if a light has been switched on. Our grandfather grasped immediately the implications, and started a long discussion about how 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 clue; the thrill of a slippery answer finally coming to you; and the satisfaction of the whole puzzle finally solved.
>
> Our grandfather understood full well that my brother's explanation wasn't meant to be taken literally. My brother was obviously not really sitting with a newspaper and a pencil solving

actual crosswords all day. Rather, he was saying that things that are true about crosswords are also true about software development. He was using a metaphor as a way of anchoring the unknown (programming) to something that our grandfather already understood well (crossword puzzles) and therefore could elaborate on.

Metaphors, then, are living things, that reduce our fear of the unknown. They are much more active than lists of prescriptive rules or heuristic values, in that they are open to continual elaboration and adaptation to changing circumstances. Whereas beginners rely on prescriptive rules to follow a path laid out by others, and intermediates call on personal heuristic values to make context-specific judgements, experts use living metaphors as generators for new rules and heuristics to bridge gaps they must cross as they face unfamiliar situations.

I revisited many of the experts whose heuristics had proven elusive, and started afresh, this time looking for their underlying metaphors.

What I discovered, unsurprisingly, is that large numbers of experts do have plenty of metaphors they called on. What did surprise at the time was that it doesn't matter much in which area the person was experienced, they would still be driven by metaphors they had cultivated. I came across many folks who had never worked in the software business, and had switched industries, and relied heavily on metaphors grounded in successes in those prior industries. This gave them certainty in the face of uncertainty.

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 came across sales people, now in charge of software teams,who negotiated agreements with teams and then treated those agreements like contracts. I worked with a search-and-rescue helicopter pilot, now a software team leader, who would swoop in on troubled projects, and try to rescue them single handedly.

As a concrete example, at one point I worked with Ann Verhoeven, a director at a well known website. Ann had previously worked as a restaurant manager. Restaurants are high stress environments, with customer service paramount to survival. I noticed that whenever there was a project crisis Ann would feel the tug of her restaurant management metaphor, telling her to keep the chaos of the kitchen (here, software developers) hidden from diners (here, customers).

When team members proposed solutions to a crisis, Ann would remain level headed and

ask whether or not customers would benefit from the result. Keep the customer happy, Ann would say, and they will keep coming back for more. Give them poor service and a shoddy product, and they will take their business elsewhere and we will be out of business.

Metaphors are particularly appealing for software development because we are all still trying to work out what the rules of the game are. Metaphors allow us to call on more established fields, where the rules of the game are more certain, thereby helping us bridge the gaps inevitably left open by the incomplete prescriptive rules laid out in methodology books. In fact, many experts have told me that, armed with compelling metaphors, they can throw methodology books away, since they now instinctively know how to handle any situation they come across, no matter how new or unexpected.

We will see in following chapters that each of the three dominant software development cultures is underpinned by specific metaphors, that sustain that culture and that are reinforced by that culture. We will also see that if you want to change cultures, you have to help people first change the anxiety reducing metaphors upon which they are heavily dependent and to which they are strongly attached.

Chapter 3 : Cultural Metaphors

Captivating Implications

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, as infants, we learn basic spatial metaphors, such as up being good, and down being bad. We hold onto and explore and exploit these spacial metaphors for the rest of our lives: “It's time we started living the high life!”; “His chances of a promotion are going downhill fast!” Later, we learn other metaphors, such as ideas having directions: “I see where you are going with this!”; “I just don't get where that guy is coming from!”. And time being money: “Doing it this way will save you lots of time”; “We have invested months in this project”. We then spend the rest of our lives 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.

The basic principle of metaphors is that concepts commonly associated with one thing, are transferred over to another. These concepts are called the implications of the metaphor. Exploring those implications helps us understand something that is unfamiliar to us in terms of something thing that we already understand quite well.

> 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 this 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!

Of course, metaphors only really work well when they have relevant implications. That is why some metaphors grab up, and others leave us underwhelmed.

> 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 nothing but air.

We can see the connection – the implications – readily. The metaphor works well.

Similarly, if somebody had talked of “The dot com gold rush”, we would probably grasp readily the implications – with images of crowds of people charging in on the speculative hopes of big profits, but with only a few big winners emerging, and most left sorely disappointed.

However, if somebody had mentioned “The dot com hamburger” we would almost certainly struggle to connect the implications of hamburgers with what happened to dot coms. Hamburgers are a pretty weak metaphor for dot com.

Unless of course, you work in the hamburger industry, in which case they may resonate profoundly.

Here, then, is a crucial aspect of metaphors. Different people can interpret the same metaphor differently since their differing backgrounds affect how they each approach, discuss, explore, reflect upon, and elaborate that metaphor. It is this openness to personal and ongoing interpretation that makes metaphors so useful, as we will soon see. At the same time, though, it means that where some people get very positive feelings from a particular metaphor, other people experience very negative feelings.

Many computer programmers smile knowingly when one says to another “My manager is Dilbert's pointy-haired boss” but their managers would likely cringe at the same thought (unless they were thinking about their own boss of course!). For metaphors to work, then, their implications have to resonate for you personally.

If the implications of a metaphor are captivating for us, we will likely accept, enjoy, and continue to explore the metaphor. If the implications are repulsive, we are almost sure to reject it and turn to other metaphors which we find more comforting.

I mentioned, to a colleague, my brother's metaphor that “Programming is solving crossword puzzles all days.” This metaphor had worked very well for my grandfather, yet my colleague seemed both shocked and repulsed by it. “No it isn't” he declared “Crossword puzzles have one right answer and programming doesn't”. The metaphor that had been so evocative for my grandfather, had simply failed for him.

My colleague countered with his own metaphor “Programming is a young man's game.” Well, that metaphor may well be appealing to a young man such as himself, but it would

undoubtedly alienate my elderly grandfather!

Culture Clashes

The metaphors somebody finds captivating and comforting help them reduce their anxiety when looking for solutions to the problems they face throughout their working day. People who find the same metaphors captivating will share a common mindset and will approach similar problems in a similar way. They will simply see eye-to-eye, and they will be drawn together. Somebody who doesn't share that mindset, on the other hand, will look at the world from a very different perspective, will have a hard time fitting in, and probably won't hang around for long.

Self-Serve asked me to help build up one of their teams. There were five open positions. We received a whopping five hundred and eighty five applications. These were whittled down to a final twenty, all of whom appeared highly capable of doing the job.

Those twenty were all interviewed by potential future team-mates and managers. Only two were hired. Two were rejected because they had clearly exaggerated their abilities on their applications. 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”.

These personality clashes came down to incompatible mindsets. Hiring any of those sixteen candidates would have been like putting a fox into the hen house. Or maybe a hen in the fox house.

The prevalent mindset shapes the culture of a group or an organization. A common definition of a culture 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. People can and do change their work quite often, but that doesn't necessarily mean they have changed their culture. Maybe a better definition for culture is "the way we see things around here". That is, our mindset, which shapes our perspective, and through the metaphors we employ, determines not just the way we work, but also the way we think and the way we approach problems on a daily basis.

Cultures clashes happen when mindsets, and therefore metaphors, collide. Too often we imagine that people will somehow adapt themselves and fit in. People can't just decide to fit in – their driving

metaphors mean 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 will have to remain anonymous had struggled with a number of projects over the past couple of years. Senior executives hoped that hiring better people would make all the difference. They 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 had a mindset that could not accept the disruption of changing current processes and policies.
>
> I know very well (ahem!) somebody who experienced that culture-clash first-hand. Having gone through a recruitment process lasting several months, he resigned within weeks of starting the job. Senior executives were not happy to lose 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 forced to fit into a conflicting culture, and this stifled the very things they could bring to the company. They were 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. Most of the new hires were gone within a few weeks. A few learned to comply: they told me they had 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. The simple message here is, don't bring in people to shake things up unless you are willing and able to take the cultural shaking.

Emotionally Charged Metaphors

Let me pause for a moment for a word of caution.

Since metaphors can be 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 people against particular methodologies or cultures in highly manipulative ways.

One issue I faced, then, when uncovering cultural metaphors is that some experts had a tendency to use highly emotionally charged metaphors when discussing their cultures with me. It took some time for me to recognize that often these were not their personal metaphors at all, but rather they were playing with provocative metaphors to convince me that their particular culture or methodology was superior and all others inferior.

I cannot blame them for doing this, since the need to convert others to our own mindset seems to be almost universal, but it certainly distracted me at times, particularly in the early days, from uncovering the software development metaphors that people really call on themselves.

I learned to be very cautious when I heard metaphors that were highly emotionally charged, and would always question them and dig a little deeper to see if people really did use those metaphors themselves, or they were just using them as a manipulative tool to persuade me and others.

Negatively Charged Metaphors

There are certainly plenty of metaphors that people can and do use to capture and reinforce negative feelings, often for cultures and methodologies that they don't known much about, and these metaphors can be pretty infectious and highly persuasive to others. For example, describing a particular methodology as a Black Hole is only going to inspire negative thoughts. The implications of such metaphors do not help people working on software development projects to address thorny problems. Instead, they are plain and simple emotional manipulation. We need to remain alert to, and guard against them.

> I heard somebody disparage one methodology by repeatedly 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 know people who become fearful of a relatively popular methodology after hearing it called the Mad Scientists Laboratory – bringing to mind images of out of control lunatics pursuing their own dangerous whims.

Positively Charged Metaphors

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

I heard a salesman constantly referring to the methodology and tools he was selling as 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 ex-managers always called a rather outdated methodology to which he was attached Proven Best Practices, yet labelled a new and promising methodology the Wild West Road to Chaos. You can guess how a senior executive at the company 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 must 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 tried to pay close attention to emotionally charged language, and the metaphors that feed it, and worked hard to steer the conversation back to more neutral language. Hopefully, I have been reasonably successful with this, and as a result the metaphors presented for various software development cultures in this and later chapters 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 – which is what draws them to different cultures - but my hope is that these reactions stem from their personal interpretations, rather than an emotional bias in the metaphors themselves.

When I first heard about the methodologies that tend to appeal to people in a Service Culture they were generally known as Lightweight methodologies. The originators of this metaphoric name 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 the word Agile is not without its problems. It is a metaphor that 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 such methodologies can indeed help projects to go quite quickly, that is not their central focus. Their real focus (at least in my experience) is on continually tracking and satisfying

customers' ever-changing needs.

As a result, I prefer the Service Culture name and metaphors for these methodologies. I have found these metaphors result in newcomers exploring associations and gaining mental images that are better aligned with those of people who already use “Agile” methodologies.

Uncovering Cultural Metaphors

As I have immersed myself in various organizations, and worked closely with people to uncover the metaphors they appeal to in their daily work, certain patterns emerged. There were three clearly dominant mindsets, each reflecting a whole bunch of metaphors, and hence each reflecting a distinct culture. The three Software Development Cultures that I repeatedly came across were those mentioned in chapter one:

- A Manufacturing Culture
- A Design Culture
- A Service Culture

These are certainly not the only software development cultures in existence. They are simply far and away the most prevalent. If and when other cultures gain mainstream popularity, later editions of this book will be updated to include them.

Of course, as well as new cultures appearing, dominant cultures can lose their popularity. In the 1950s, for example, none of the three currently dominant cultures was widespread. Instead, the dominant culture was the Science Culture. This 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.

Unfortunately, Software development 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. These game-rules seemed to not be working. This shook confidence in the Science Culture, and many people, particularly industry, began to abandon it.

Still, cultural ties can be pretty strong, and even to this day some people – despite all the

evidence – cling to the Science Culture, and typically blame all failures on a lack of abiding by the rules upon which this culture is built. For those who are interested, the Science Culture is explored in greater detail in Appendix A, but we will pursue it nor more here since it has mostly fallen mostly into oblivion.

Each of the three software development cultures is underpinned by very different metaphors, that bias people's perceptions and judgement. Just the names alone are highly evocative metaphors.

Think of Manufacturing, for example, and all kinds of thoughts probably spring to mind. Maybe you visualize highly skilled workers operating advanced robots working in silent efficiency to assemble high technology components.

Think of Design, and maybe you imagine talented individuals sitting in quiet contemplation, while they consider the various dimensions to a problem, homing in on a single elegant design.

Think of Service, and maybe you see professionals working closely with their clients to uncover and satisfy their particular needs

Or maybe not.

Maybe Manufacturing conjours up for you images of noisy factories, with sparks flying, and sweaty workers feeding raw materials into heavy machines that hammer things into shape and then spit finished goods into a heap on the factory floor.

Maybe Design has you thinking of noisy brainstorming sessions, with prolonged and heated arguments over all kinds of weird and wacky design proposals.

Maybe Service has you cringing with repulsive images of slaves being beaten into submission by their evil masters.

The cultures that have one set of connotations for some, have very different connotations for others.

Dominant Metaphors

Beyond its name, I have found that each of the three cultures is reinforced by a single dominant, driving, metaphor:

- **Manufacturing Culture**: Software development is a factory production line
- **Design Culture:** Software development is architectural design
- **Service Culture**: Software development is customer satisfaction

Each of the three dominant metaphors colours our judgement and guides, or blinkers, our lines of thinking in deeply emotional ways. Anything that matches the dominant metaphor of the culture 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 dominant 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.

> A salesman I came across seemed to view software development 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.

Elaborating a culture's dominant metaphor helps people committed to the same culture frame a shared mindset for what software development is really all about. Over the past few years, I have worked closely with many people across each of the three dominant cultures, and have asked each of them to summarize briefly their mindset with respect to their own culture's dominant metaphor. Below, I hope I have captured reasonably faithfully the general flavour of the patterns in those summaries:

- **Manufacturing Culture**: Software development is a factory production line

  Factories have already streamlined the most efficient way of producing products, which is to use production line. Since software is also a product, it too will be most efficiently produced on a (virtual) production line.

  The smooth operation of a production line requires up-front planning by managers to prevent nasty surprises such as time and cost overruns, or poor quality products. The software requirements must be pinned down in that up-front plan so that workers have a clear specification for the product they are to manufacture. Quality assurance staff will use that same specification to measure the quality of products at the end of the production run.

  Workers are assigned in sequence to individual stations along the production line. The worker at each station is responsible for specific and well-defined tasks as detailed in the plan, such as requirements specification, detailed design, programming, testing, etc. Each such task is to be

completed within a pre-planned time limit, which in aggregate gives the overall time-schedule for the complete production run.

Each worker on the production line receives the output of the worker before them, completes their own tasks as described in the plan, and passes their results along to the next worker on the production line for further work. Completed software then rolls off the end of the production line on budget, on schedule, and matching the specification.

Managers must monitor workers closely to ensure that they adhere closely to the plan, and re-steer or punish those who do not do so, since deviating from the plan is the greatest threat to project success.

- **Design Culture:** Software development is architectural design

  Architecture is the conceptual scaffolding around which products are designed and built. Great architectures are designed to anticipate users' changing needs, and therefore have flexibility built right in.

  Great architectures are created by great architects, who are experts at clarifying product requirements and making scale models in order to investigate uncertainties, to address tricky problems, to experiment with promising design solutions, and to anticipate and prepare for future changes. Throughout this design process, the architect identifies points of architectural stability, and builds-in place-holders for anticipated future change. Once the architectural design is stabilized, the rest of the product can be built on it with confidence.

  Software is a product designed to support a user's business processes. Software designs that do not anticipate the future business process changes that users will face are fragile. They break under the strain of such changes, and soon lose their business value.

  Flexible software, on the other hand, is built around an architecture designed with such change in mind. Flexible software architectures, then, anticipate future business process changes and are designed with built-in place-holders in preparation for them. This ensures that the software can be adapted quickly if and when those business process changes do occur, and therefore will continue to remain highly valuable to users.

- **Service Culture**: Software development is customer satisfaction

  Customers want to do business with us when we help them satisfy their most pressing business needs. This means focusing primarily on *their* success rather than only focusing on our own.

Businesses today are in continual competition in their markets. To survive and thrive they have to keep responding innovatively to unexpected moves by their competitors, and find newly emerging opportunities whenever they can. Success, therefore, requires the on-going adaptation of their own business processes in ways that cannot be anticipated and prepared for in advance.

Since business processes change over time, so must the software that supports them. Software must remain forever soft, and its design needs to adapted continually, in response to such unanticipated changes.

Customers should be granted a central role in identifying their shifting business priorities, and determining which software features should be developed, and the order in which they are delivered. Delivering fast and often allows customers to use product features while they still have high business value. It also allows them to change priorities for which features are delivered next, and halt further development when cost outstrips potential value returned.

Customer satisfaction, then, means shifting our software development focus away from our own desire for stable software designs, and instead serving the customer's need for the software to continually support their changing business processes. We then become a real asset in the eyes of the customer rather than merely a business expense.

Consequent Metaphors

The dominant metaphor of each culture, then, helps people form the general mindset shared by people committed to that culture. With that mindset firmly in place, they are then able to continually elaborate their dominant metaphor to help them frame all sorts of issues, covering some pretty fundamental aspects of software development. Most notably:

- Product related matters such as what software, design, and requirements are all about
- People issues such as the roles of managers and teams, and their relationships with customers
- Risks that can affect project outcomes, particularly concerns over scope, time, quality, and money
- Measurements that dictate what counts as progress and what counts as waste, and that also determine whether a given project should be considered a success or a failure

With these issues in mind, you can play a game of word association, where you raise the issues one by one, and ask people for their immediate instinctive reaction. For example, say to somebody “Success

is ... ?” and await their reply. You will find that their instinctive response almost invariably matches an implication of their own culture's dominant metaphor.

Naturally, different people from the same culture are going to give different response. For example, when I have said “Money is ... ?” to people from a manufacturing culture their responses have varied dramatically. Some have said “power”, others have said “profit”, others have said “spent”, and so on. I faced a major challenge, then, when trying to find patterns in these responses, but over time I have generalized and refined them as best I could. The table below is my own humble attempt at summarising the instinctive reactions that members of each culture lean towards.

	Manufacturing Culture	Design Culture	Service Culture
Dominant Metaphor			
Software Development is	A Factory Production Line	Architectural Design	Customer Satisfaction
Product Issues			
Software is	Hard	Flexible	Soft
Design is	A Blueprint	A Technical Activity	A Human Activity
Requirements are	Captured	Modelled	Shared
People Issues			
Managers are	Commanders and Controllers	Leaders	Referees
Teams are	Workers	Experts	Collaborators
Customers are	Buyers	Business People	King
Risk Issues			
Scope is	Fixed	Clarified	Fluid
Time is	Money	Quality	Change
Quality is	Checked	Stability	Customer Acceptance
Money is	Cost	Productivity	Valueless
Measurement Issues			
Success is	Compliance	Well Designed Products	Happy Customers
Failure is	Punished	Spaghetti Code	Detected Early
Waste is	Sloppiness	Bureaucracy	Low Value
Progress is	Task Completion	Design Improvement	Continual Delivery

What is particularly striking in this table is that many of the implications listed are themselves clearly metaphoric. They are consequent metaphors of the culture's dominant metaphor. When somebody says “Software is Hard””, or “Money is Productivity”, or “Time is Change” it is not meant to be taken literally. Even those which at first appear literal, such as “Quality is Checked”, “Design is a Technical

Activity", and "Failure is Detected Early" are metaphoric to members of their respective cultures, since each triggers further ongoing mental activity, leading to the discover of fresh insights as its own implications are explored.

> The table has reached the point where, after many iterations, I am reasonably confident that it is an accurate reflection of the implications of each of the three cultures. Or, to be more precise, an accurate reflection of the perspectives of the many people I have worked with who were themselves immersed in those cultures. It has benefited then, not primarily from own efforts at synthesis but, much more importantly, from repeated feedback from numerous people within each culture. To them, I am truly grateful.
>
> Having said that, if you have been immersed deeply in one of these three cultures yourself, and you feel strongly that I have grossly misrepresented the implications of your culture in this table, then please do contact me so I can correct any mistakes in future editions of this book.

Each culture's dominant metaphor, then, provides a context within which its consequent metaphors are framed and elaborated. Saying that "Waste is Sloppiness" relies on the context of the Manufacturing Culture's Factory Production Line metaphor to help frame and elaborate what sloppiness is. For example, disobeying orders, only partly completing tasks, spelling mistakes in paperwork, etc. A Design Culture, on the other hand, would see very different implications for sloppiness (e.g. inflexible architectural design) as would a Service Culture (e.g. lack of care for customer satisfaction).

> Earlier this year I worked with a company that wanted to offer to users much richer search capabilities across their database. The team assigned to the project saw it as wasteful to dive into concrete design until they knew which things were important to users. So, they placed a strong focus on determining realistic user scenarios.
>
> After two weeks they made a presentation to an Executive Vice President, and I was lucky enough to sit in on this. Each time they presented a user scenario, the EVP would jump in with very detailed user interface questions: "Will the search results be on the right or the left?", "What color scheme will be used on the search criteria box?", "Will this be a check box, or a radio button?"
>
> The team explained they had focused on identifying the most valuable requirements rather than waste time on premature design. The EVP seemed exasperated. He cut off their explanation and told them he expected to see a very detailed plan, not time waste time on "wishy-washy" matters. They were given another week to go away, come up with a project

plan, including a time schedule, detailed task list, and user interface decisions, then come back with the presentation that “you should have been giving today!”.

Understandably, the team went away muttering that they would be wasting a whole week on this presentation, and the EVP went away frustrated that the team had just wasted the last two weeks.

Metaphors and Methodologies

The real richness of metaphors, then, comes not from their immediate implications, but rather from the very personal way in which we can keep on elaborating them in unforeseen circumstances and at varying levels of complexity. A culture's dominant metaphor, then, can grow with us as we gain experience elaborating its underlying consequent metaphors in new and varied situations.

This process of personal elaboration invariably leads to new internal metaphors, weaving together new concepts, new vocabulary, and new relationships, which themselves are subject to a personal and ongoing journey of recursive elaboration. These continual discoveries are the “Aha!” moments, as the metaphors guide you to fresh and exciting insights.

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 him 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 off the top. New metaphors were clearly unfolding in his mind, and it was delightful to see his excitement grow as he unravelled them personally.

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 software development novice into a development expert.

When we stop elaborating a metaphor, it is essentially dead. In fact, that is a good definition for a methodology expressed as a rule book: it is the resting place for the scattered bones of dead metaphors

that were once elaborated by its author. The loss of the underlying metaphors from which those rules sprang leaves us without the rich body of implications to draw on and further elaborate. We have no clue as to how to put flesh on the bones. We are left following slavishly the prescriptive rules of the methodology, which are the only remaining traces of the authors own frozen elaboration.

> A company that publishes statistics hired me for two months to help a senior software architect with some particularly tricky design work. As I learned about requirements, I opened an editor, and started typing in some code: a small test, capturing a small requirement. The architect warned me that if the head of the department saw me, I would be reprimanded severely. But surely, I said, real architects of buildings create scale models and test them. That was irrelevant he replied. The methodology book said that 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 their methodology seemed pretty crazy to me, since it meant the architects had no way of knowing that their designs were any good.

For a methodology to remain adaptable to changing circumstances, its underlying metaphors would have to remain evident. People can then continue elaborating them further as opportunities arise and needs dictate. Such methodologies then remain living things, rather than long-dead corpses.

> This, I have found, is precisely how experts treat even rule-book methodologies. Several experts told me that once they had waded through the rules they could assimilate the essence of a methodology. They would gain a kind of instinct for where the methodology was headed, and could then throw away the rule book, because they no longer needed it.
>
> This instinct obviously didn't come from remembering a long list of of rules. Rather, I discovered that the experts had the ability to abstract from the detailed rules and form metaphors consistent with the methodology that helped them feel out missing rules for themselves.

A methodology which is expressed in terms of its metaphors is open to continual elaboration. Rather than being a graveyard for scattered bones, the methodology forms a skeleton – a conceptual scaffolding - which is gradually fleshed out over time. The methodology writer may start us out with a single, highly literal, interpretation in terms of a first set of rules. Vitally, though, the metaphors tell us how to handle gaps in these rules. We don't have to look in vain for concrete answers the rule book does not contain, we can call on the methodology's underlying metaphors to help us add flesh to the bones.

Chapter 4 : Manufacturing Culture

The Factory 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.

Other industries faced similar problems in the past, and they worked out how to deal with them. The mass manufacturing industry in particular has led the way, and established the production line as the most streamlined approach to ensuring that projects can be on time, on budget, and deliver high-quality products regularly and predictably.

Mass manufacturing's principles of the production line caught on in other industries, where they were recast as the basic principles of professional project management, and they again proved themselves highly effective.

It should come as no surprise, then, that many people involved in software development have been drawn to a Manufacturing Culture and its dominant, driving, Factory Production Line metaphor to help them tame their own wild projects.

The Chocolate Factory

Visualize in your mind's eye a clean and well-maintained factory, producing boxes of orange-cream-filled chocolates reliably and efficiently. Notice the production line, with chocolates laid out at increasing stages of completion along the conveyor belt. Look at the machines stationed at various points along the production line, and see the workers operating them.

Now run the production line like a movie. Add colour, and smells and sounds if you can. Notice what happens as the operator at the first machine presses a button, and pulls a level. The machine at that station creates empty chocolate shells and lays them out on the conveyor belt. The chocolate shells pass smoothly along, and pause at the next station. The operator at that station turns a handle, and each chocolate shell is filled with just the right amount orange cream. The chocolates move forward along the production line, and pause as the next operator flips a switch upwards, and a machine places a walnut on top of each chocolate. The chocolates move on to the next station. The operator presses a pedal and the last machine on the production line machine picks up the chocolates ten at a time, puts them into neat rows in boxes, and places them in racks for shipping.

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

Yikes! Look out! One of the machines on the production line has broken 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? Come on, we need a decision! Now!

As an experienced production manager you have faced this situation before. You know how to deal with it. 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. Job well done!

The Luxury End of the Market

It is six months later. You have kept the production line running, established standardised procedures and emergency measures. Things run smoothly when you are in charge. You have a well-earned reputation as a great production manager. You are head hunted by a more prestigious chocolate company that specializes in a luxury range of hand-made chocolates.

Picture in your mind 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 few chocolate shells 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 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 place 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. They decided to go off to a nearby cafe for twenty minutes rather than just sit at the table and wait.

Another artist is adding final decorations to a handful of chocolates. His chocolate knife slips, consigning two more chocolate to the misshapes bin. He places the surviving chocolates one by one into individual boxes, and ties each one by hand with a ribbon. When twenty boxes are filled, he moves them over to a tray one by one, and takes them over to the shop front in the next building. Ten minutes later, he is back, but stops off for a smoke, before returning to 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 couldn't be done more efficiently. The 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 is subject to the whims of highly-skilled and highly-paid chocolate artists. You are sure things could be run much more smoothly than this.

By the end of the day, the twelve artists have made eight hundred and seventy three chocolates. Less than the one thousand per day target. Production is your responsibility now, and you are held accountable if targets aren't met. Still, it is early days yet. Hopefully, the chocolate artists will make up for the shortfall tomorrow.

A couple of days later, the sales director grabs you. He is in a real panic. A major competitor, keen to dominate the luxury end of the chocolate market, has held a press conference. They have reduced dependency on chocolate artists by automating much of their work. They now employ low-cost machine operators, making high end chocolates on a production line, at an eighth of your cost, with waste reduced by seventy five percent, and production volumes twenty times higher than your artists could ever manage. If you don't respond, there is a good chance your company will be out of business in months.

The competitor's chocolates 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.

As production manager it is your responsibility to sort this out. Senior managers are putting you under intense pressure. They want an answer! You are either up to the job, or you will be out of a job! What are you going to do?

A large 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?

You know what needs doing, and you do it. Within three months you have completely moved the company over to a factory production line. Within six months it is profitable, and within nine months your company is dominating the market.

The Software Factory

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 seem 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 handmade chocolate company was run before you injected more discipline, predictability, and cost control. 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 exactly the same 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?

All your experience tells you to run software projects on a factory production line. It has proven itself time and again in the chocolate factories. A smoothly run production line is the difference between predictability and chaos. You can do this, you can bring costs down, you can bring order to these projects. And so, that is exactly what you do.

From Craftsmanship to Mass Production

Your experience in chocolate factories drew you to a Manufacturing Culture, along with the assumption that it doesn't matter much if you are producing airplanes, spoons, chocolates, or software. Projects run smoothly by applying the basic principles of production line management that were first established in mass manufacturing decades ago.

Before production line management principles were established, there was a great deal of reliance on craftsmanship. 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. That is exactly how the high-end chocolate factory worked before you turned things around, and it seems eerily like the way the software company had been running until you arrived.

With mass production, on the other hand, workers stay stationary as the product moves in front of them along a production line The specification for the product is presented in an up front plan, along with a breakdown into stages of the total work required to produce it, and a time schedule and monetary budget for those stages and for the production run as a whole. Each worker performs the tasks defined for their stage of production, and passes their output along to the next worker on the production line for further work. Completed products roll off the end of the production line on budget, on schedule, and matching the specification. Managers ensure that workers adhere closely to the plan, and punish those who do not, since deviating from the plan threatens project success.

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.

Project Management

The success of the production line in mass manufacturing was inspirational for managers in many other industries. For too long they had witnessed projects going haywire: exceeding their budget; dragging on for too long; missing important product features; or being of variable and shoddy quality. They looked to control such risks and inject more predictability into their projects.

They called on the production line as a metaphor to show them the way. They 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 a template for project management whose steps have come to be regarded as best practices, even standard practices, for managing any project irrespective of industry. Plenty of courses are now available that show you exactly how to repeat these steps for your own projects. Certification is available that recognizes your professional project management competence.

Here are the basics, all grounded in the proven efficiencies of the production line:

Plan ahead:

- 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, and produce a corresponding time schedule for each stage, including a start and end date for the project as a whole
- Determine the economic resources (raw materials, cost of labor, machines, tools, etc.) required for each task. Sum these to determine the cost for each stage and the overall project budget

Start Production according to the plan:

- 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 to begin the production run

Monitor and Control:

- Continually track:
    - Workers at each station to see if they are performing their tasks according to the plan
    - 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
    - The finished products at the end, checking that they meet the quality standard
- Where inspections find problems, regain control:
    - Stop the production-line.
    - Rectify the cause of the problem where possible by:
        - Fixing broken equipment
        - Shouting at and threatening the workers
    - 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

Software Development is ... a Factory Production Line

By the 1970s the software industry started to take notice. Other industries had long since demonstrated that individual craftsmanship, upon which the software industry was still heavily dependent, brought with it chaos, uncertainty, and risk. Whereas the production line, and the administrative management that monitors and controls it, had proven extraordinarily effective in taming project risks and brought order, predictability, and control.

Many software project managers were inspired by this success and sought to reduce their own project risks and inject greater predictability, reliability, and discipline. As a result they were drawn to a Manufacturing Culture and the factory production line as its dominant metaphor. They created virtual software factories with simulated production lines monitored and controlled via proven project management principles. Typically:

- Managers create a project plan which breaks the necessary work down into a series of stages, with strict time and cost budgets set for the various project stages, and hence for the overall project.
- Workers are taken from the staff pool and assigned in sequence to the various stages of the project:
    - Requirements Stage: A product specification is captured up-front by the customer, or another authorised person, translating their business needs into a business requirements specification document.
    - Analysis Stage: An Analyst translates the business requirements specification document into a complete and fixed set of functional requirements for the software, producing a functional requirements document
    - Design Stage: A Designer translates the functional requirements document into a rigid structural blueprints for the software, forming a technical design document

    - Coding Stage: A Programmer follows the blueprints contained in the technical design document to produce the required software code.
    - Quality Assurance Stage: Testers examine the functionality of the software code to ensure it meets the specification detailed in the earlier documents
- The workers at each stage provide periodic status reports so that managers can monitor and control their progress

Overview of Consequent Metaphors

The previous chapter mentioned that the production line metaphors lead us to numerous consequent metaphors. These will be explored in great detail in later chapters. For now, though we make do with a brief overview of them, to give a general flavour of the mindset of people committed a Manufacturing Culture.

Software is ... Hard

Software is treated like a hard physical material, such as metal, that get bashed into its final shape according to a set procedure, producing the finished product. The pinned-down up-front requirements specification determines the shape that the software should take.

Now, manufacturing production-lines require the physical item in production to be passed forward from stage to stage. There is, of course, no such physical item in software development; the closest would be the software itself, but this is non-existent until late on. Therefore, documents substitute for that hard physical item until the software coding stage beings.

Once the coding stage has finished, the software should not need to change, other than for occasional maintenance work to keep it running.

Design is ... a Blueprint

The technical design document produced by designers is a detailed blueprint for the programmers to followed closely and accurately when creating software. Since each document in the production process is an accurate translation into a new format of the document that preceded it, everything specified in the design blueprint is ultimately traceable back to a requirement agreed contractually with the customer. As a result, there are contractual obligations that demand programmers comply fully with the detailed design laid out in that blueprint.

Requirements are ... Captured

There is a frequent fear that a worker will be hit by a bus or leave the company, taking all their project knowledge with them. Rather than allow workers to carry things around in their heads, they are required to capture the results of their work completely and accurately in documents. The documents produced at each stage on the production, then, line capture all project progress, so that nothing is lost when workers are replaced. This makes documents the primary and most reliable means of communication. Each document fully captures all that needs to be known by the worker at the next stage of production, and should be seen, therefore, as the sole requirements document upon which the worker at that next stage bases all their work.

Managers are ... Commanders and Controllers

Expecting workers to make arbitrary decisions throughout their project work means that nobody is quite sure what is going on at any point in a a project. To reduce this risk, planning is used to make as many up front decisions as possible. Where further decisions are required during a project, managers have the necessary authority to make them, and they are then responsible for ensuring that workers carry them out. Managers can be thought of as commanders, rather like in the military, and they continually monitor and control workers to ensure their commands are being followed correctly.

Teams are ... Workers

The individual workers assigned at each stage of production take a document written in one format, translate it accurately and on schedule into another format, and pass it on to the the workers at the stage for further translation, until the final translation (software code) falls off the end of the production line on schedule, on budget, and matching its original specification.

The only skills required are knowledge of the language formats you are translating from and to and speed and accuracy in performing that translation. For example, designers do not need to understand business concepts (that is a concern for analysts) nor do they need to understand programming languages (that is a concern for programmers) they merely need to be efficient at translating functional requirements into structural blueprints.

Managers drawn to the production line metaphor often ask if the whole translation process couldn't be totally automated. Get the computer to do all the translation, so that business people can just type their requirements into the computer directly, and code shoots out of the other end, thus eliminating the need for and expense of translators along the production line.

Customers are ... Buyers

The software provider is a supplier of products, and the customer is the buyer of those products. The business requirements document forms a contract between the customer (so they know what they are getting) and the software supplier (so they have a clear target to focus on). Once this document is signed off by all parties, each party is then obliged to live up to the contract. The software supplier is obliged to produce software exactly as specified in the contract, and by a date agreed in the contract. The customer is then obliged to pay the contracted price for that software.

Scope is ... Fixed

There is no point starting a project if the nobody knows its goals, since it will never get anywhere. The goals of a project form its scope. Changes to the scope of a project are strongly discouraged mid-project, since this introduce considerable risks. The scope of the project is, therefore, fixed up front in the business requirements document, and it then considered frozen for the duration of the project.

Time is ... Money

Since there are few raw materials involved in the production of software, the primary expense is worker hours. Excessive hours spent by workers on their tasks leads budget overruns. To control costs, the time spent on each task must be optimized up front, scheduled in the project plan, and monitored and controlled during the production run.

Quality is ... Checked

Since each stage of production is merely a translation of a document into a new format, it is relatively simple to ensure that the output of each stage is of good quality. All that needs to be done is for managers and authorised workers to review a documents, check that it is an accurate translation of the prior document. and sign it off as being of sufficient quality. The software itself can be thought of as a final translation of the whole series of documents that preceded it. Quality assurance for software is then ultimately a matter of checking its delivered functionality against that specified in the contractual obligations that were agreed with the customer (and were captured and signed-off in the original requirements document).

Money is ... Cost

Businesses exist to make a profit. Profit is sale price minus production costs. Once the sale price has been negotiated with the customer, any reduction in production costs will increase profitability. Money spent on production, then, must be streamlined, budgeted for up front in the project plan, and closely monitored throughout production using well established cost accounting principles.

Success is ... Compliance

This type of work doesn't require creativity, it requires compliance. So long as all workers follow the plan correctly, and do as they are told, the results will be as expected. A project, then, must be driven as closely as possible to its plan, to ensure that each stage starts and finishes on time and within budget and the finished product matches its specification. When everybody complies with their responsibilities as defined in the plan, the project will run smoothly, and there will be no unpleasant surprises. Success, then, is measured in terms of how well the plan was adhered to.

Failure is ... Punished

When people miss firm deadlines, or produce shoddy goods, or fail to complete tasks as laid out in the plan or as instructed by their managers they endanger the whole project. When workers fail to live up to their commitments, they must be held accountable. Managers, then, have the authority to punish them. Punishments may be a simple as a reprimand, or may extend to loss of a bonus, or even in severe cases warrant loss of their position in the company. Awareness of these potential punishments should serve as a deterrent factor, ensuring that employees will work hard to remain on the straight and narrow.

Waste is ... Sloppiness

Each stage of production is carefully planned to produce the correct output efficiently and with a minimal amount of waste. Waste results from being sloppy in adherence to the streamlined plan. Sloppiness lead to rework, and rework increases resource consumption, thereby risking budgets, set time schedules, and quality standards. Not only does this impact profitability, it also increases the likelihood of being unable to meet obligations laid out in the contract with the customer.

Progress is ... Task Completion

Influential experts in the software world have long ago argued convincingly that mistakes made in requirements or design are expensive to correct once programming had started. Therefore, production-line software development insists there can be no going backwards to ask for clarifications or make changes. Progress can only move forwards. Therefore, all tasks at each stage must be completed fully and accurately.

Status reports enable managers to track the progress and completion of tasks within the current stage on the production-line. There also needs to be a way to signal that a given stage is now complete, and the next stage can begin. For physical items, it is easy to recognise that a machine has finished some transformation of the raw material. For software production it is more difficult. One approach is for the analyst or designer to say “I am done”. However, many companies prefer to leave such important

decision to managers, and hence often have managers review and then sign-off documents, declaring them error free and frozen. This is the official seal of approval for moving forward to the next stage along the production line.

This forwards-only principle has led many people to describe the production line as a waterfall approach to software development, since water flows down a waterfall and never flows back up.

Chapter ? : Teams are ...

What is a Team?

In the earlier chapters 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 doesn't just affect individuals, it also affects the relationships people form, the work they do together, and how they measure their own and each others success. That is, beliefs, values, and metaphors are at the heart of teams and teamwork.

Of course, everybody talks about the virtues of 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.

NEED AN EXAMPLE HERE

You can certainly assemble a group of people, then, 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 unite, gel, and flourish

How Does a Team Unite?

A team, then, 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, nobody can 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 team members are kept apart, and deprived of opportunities to develop a shared sense of belonging, they will never develop a common identity, and the team will not unite. Likewise, I have seen that deliberately breaking up a team at the end of a project and returning its members to a “resource pool” destroys 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 group of people 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, which are the rules they agree to play by as they work together. There can, of course, be an internal struggle of competing beliefs before any dominate. The founder of the team or a particularly dominant team member can even impose their own beliefs on the team, particularly if they have authority over the team, but these beliefs will later be put to the test.

During my childhood, I played a game called Cowboys-and-Indians with other boys in the neighbourhood. We would act out the major battles we read about in adventure books. One boy, General Custer, owned most of the toy guns, and lent them out to others.

At the Battle of the Little Big Horn, General Custer refused to die. He claimed he was wearing a magic shirt that repelled all bullets. When other boys protested, General Custer took back all his guns, and stormed off home, thereby ending the game.

General Custer was playing by different rules from the rest of the boys, and this stopped us from re-enacting battles properly. We soon got fed up with this, and conspired against him by making our own guns from wooden sticks and rubber bands. General Custer had to play

by our rules now, or not play the game at all.

Once a team settles on a set of shared beliefs, they can begin to form cohesive bonds which help the team gel. 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.

Gelling is important since it compels the team members to not just care about their own future but also that of the team. When a team that has gelled detects a threat, 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.

When the threat comes from within the team itself, I have seen several teams eject individual team members who threatened internal 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

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.

A gelled team, then, will have little internal friction and will run relatively smoothly. However, put in somebody new who resists the team's beliefs, 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 brought with him a strong command-and-control mentality that was entirely disruptive to the team, which had thus far relied on collaboration and consensus.

Jacques couldn't understand why the rest of the team was so undisciplined, and he 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 trying to force them to do things that distracted them from working well together.

The clash in beliefs 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. Jacques was sent packing, and projects started to improve.

In an extreme case, I have seen a team protect itself by going as far as 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. That means protecting itself from external threats.

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.

> NEED AN EXAMPLE HERE

Survival for a team, then, is dependent upon satisfying environmental needs through their daily work. If the team members see 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 external pressure to fragment.

> NEED AN EXAMPLE HERE

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 new beliefs (leading to new values) that lessen team anxiety and increase the ability of the team to thrive.

NEED AN EXAMPLE HERE

When a team overhauls its beliefs to better satisfy its environment it reduces external threats. The flip side is that belief overhauls can at the same time threaten the team's internal bonds. Team members may disagree about what the new beliefs should be, and may not have equal commitment to whichever new beliefs dominate. This threatens team cohesion and increases the chance of splinter groups forming.

It took me a while to see how long-lasting gelled teams manage to survive this threat. The answer, at least in the many teams I have worked with, is that they have mechanisms already in place for handling situations not covered by their current beliefs and values. More specifically, gelled teams become experts at handling gaps in their rules in exactly the same way as individual experts: by calling upon and continually elaborating metaphors.

Shared metaphors, I believe, form the basis of team cultures. Flourishing, then, means that a team has established a culture whose underlying metaphors guide that team to adapt and survive in it changing environment.

NEED AN EXAMPLE HERE

Once a team has established a culture which helps the team survive for the long-haul, the main threat to that team is then any heavy-handed pressure that tries to force them to drop their culture in favour of another one. The instinctive reaction of the team will be to resist all such pressure, unless their anxiety can be reduced so far that the team feels compelled to make that culture shift themselves.

Chapter ? : Managers are ...

Management Authority

We saw in earlier chapters that projects have a better chance of running smoothly when the project team has united around a culture grounded in shared metaphors. These metaphors inform the team how to behave in familiar circumstances and how to fill-in-the-gaps and react in unfamiliar situations. Even when a team has gelled around a shared culture, thought, they can still be stifled by their manager.

A manager is somebody who has authority over a team. A manager can use that authority to impose their own values on a team that instinctively resists them. With those values, the manager implicitly imposes the metaphors from which they sprang, and the culture those metaphors support. The result is a culture-clash, and the team will experience great anxiety and a strong urge to bypass and work-around the managers demands, and follow instead their own culture's metaphors to reduce their anxiety and improve the chances of project success. The manager, on the other hand, will likely see the team as endangering the smooth flow of the project and undermining his or her authority. The manager will feel anxious too, and will push back at the team.

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 member of 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 to customers until it was finally “done”.

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.

FIFO Culture

I have found that it is very common for teams and their managers to be committed to conflicting cultures. Culture clashes create human conflicts. It is no surprise, then, that we often hear folks in software teams complain that “we've got too many managers 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 is rarely because they are spoiled children who want to have fun without responsibility. It is more usually because, quite frankly, a lot of dictatorial management really is detrimental to project results.

> I had an interesting discussion with the owner of a medium sized software company, who wondered how to boost morale and productivity. After speaking to various people around the company, it was clear that the owner was really more interested in obedience than actual results. Plenty of people in the trenches had some great ideas about improving the culture to boost morale and be more successful in projects, but they were scared to air them for fear of being labeled “stirrers”.
>
> In my naivety I reported some of these cultural differences to the owner, and he shouted back: “Look, we have a FIFO culture around here. People can either Fit-In or F***-Off”. And that is exactly what people did.

Hostile Managers

Perhaps surprisingly, a team does benefit from people in hostile relationships. Hostiles provide the team with enemies, and enemies help the team unite and focus their efforts. An enemy gives the team somebody to rally against, thus strengthens the team's internal bonds. A competitor in the marketplace makes an excellent enemy, by preventing the team becoming complacent and forcing them to be creative and adaptive in order to survive and thrive.

What is not beneficial, though, is for the manager to take on that hostile role, since the manager will then become the team's enemy.

A manager is a hostile 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 culture and damage the team's internal cohesion.

In extreme cases this may well be a deliberate act of sabotage, but more often the manager has good intentions and is actually trying to help the team be effective. An alarming instance of deliberate

sabotage 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. This will certainly boost the manager's own ego and status, but at the expense of 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!"

The unwitting saboteur, on the other hand, does not realize that their impositions are actually hindering the effectiveness of the team and even threatening the team's existence. There are four classic classic cases of unwittingly hostile manager that I have seen time and time again:

1. Micro Managers 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.

   > NEED AN EXAMPLE

2. Bureaucratic Managers put complete faith in an authoritarian hierarchy and enforce 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?"

> NEED AN EXAMPLE

3. Intermediary Managers doesn't so much tell the team how to do their work, but prevent the team from forming good relationships with their environment. They attempt to “protect” the team and the environment from one another, acting as their go-between, believing that neither is well suited to interacting directly with the other. This prevents customer needs from being directly communicated and 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.

4. Indecisive Managers keep 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

Hostile managers leave a team with three choices:

1. Rebel : 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 the manager using their authority over the team to hand out some form of punishment, to ensure greater compliance in the future.

2. Comply : Give up hope of cohesion and do as the manager demands. They have then switched their focus from doing what they deeply believe ensures project success, 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 much greater with their wilful cooperation.

3. Disband : 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 hostile relationship.

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.

Supportive Managers

A hostile manager, then, 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 hostile 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.

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 their manager has little confidence in them, and treats them with suspicion, the relationship will be damaged. This rubs off on employees, who grow suspicious of the manager's intentions, losing any confidence they had in the managers, and they soon any loyalty they felt. 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. A vice president fired people from Bjoern's team as a cost cutting measure, then piled plenty more work on the team. Bjoern 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

Supportive management, on the other hand, establishes 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, of course, 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 to 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

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.

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.

Teams

Managers

What is a Manager?

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 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.

The alternative to hostile management, though, isn't no management, but rather supportive management, that help them succeed on project, rather than sucking them into time-and-energy sapping power struggles.

Supportive management 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.

A team faced with a hostile manager will likely see the manager as incompetent, and wish he or she did not interfere in such ineffective ways., and therefore meet resistance rather than cooperation. It is, therefore, no surprise teams don't want them.

A manager in a supportive relationship with a team will get their full and wilful cooperation. 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.

Manufacturing relies on Hostile Power

Design relies on Expert Power and Leadership

Service relies on manager as Referee

Chapter 13: Corporate Culture

Culture Clashes

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.

YOU CAN END UP BASHING YOU HEAD AGAINST THE WALL SINCE OTHER PEOPLE JUST WON'T CHANGE. IT CAN BE DEEPLY FRUSTRATING WHEN YOU ARE SURE YOU KNOW WHAT IS GOOD FOR THEM. BUT THEY CAN FIND IT EQUALLY FRUSTRATING THAT YOU ARE TRYING TO FORCE THEM OFF THE STRAIGHT AND NARROW AND INTO THE PATH OF DANGER.

PEOPLE ARGUE AT DIFFERENT LEVELS OF THE METAPHORS. THERE IS LITTLE POINT ARGUMENT ABOUT IN WHAT WAY TIME IS QUALITY WITH SOMEBODY WHO DOESN'T BELIEVE THAT SOFTWARE DEVELOPMENT IS ARCHITECTURAL DESIGN. SOMEBODY, THEN, WILL REJECT YOUR METHODOLOGY OR CULTURE OUT OF HAND IF IT DOES NOT RING TRUE AT THE LEVEL OF THEIR DOMINANT METAPHOR. IF THEY SHARE THE SAME DOMINANT METAPHOR AS YOU, THEN THEY MAY REJECT WHAT YOU ARE SAYING AT A LOWER LEVEL, BUT AT LEAST YOU WILL HAVE SOME POINTS OF COMMONALITY IN YOUR CULTURE AROUND WHICH TO FRAME THOSE DISCUSSIONS.

WHEN YOU CHANGE METHODOLOGIES YOU ARE IMPLICITLY ENFORCING THE CULTURE FROM WHICH IT ORIGINATED. IF THAT CULTURE CONFLICTS WITH THE CURRENT DOMINANT CULTURE YOU WILL FACE RESISTANCE, SINCE ALL OF THE IMPLICATIONS WILL CONFLICT TOO. CHANGING METHODOLOGIES, THEN, IS NOT JUST A MATTER OF IMPOSING NEW RULES, BUT OF SHIFTING THE CULTURE TO THE METAPHORS IMPLICITS IN THE METHODOLOGY.

Forcing onto a team a methodology which reflects a different cultural mindset, then, will create conflict across a whole range of dimensions, and understandably meet considerable resistance.

As we shall see in later chapters, changing culture does not mean changing behaviour. If you just do that, the tug of the prevalent cultural metaphors will either ultimately drag behaviour all the way back to where it started, or cause such internal conflict that progress grinds almost to a halt. For cultural changes to last, a whole range of cultural assumptions need changing, and that means shifting people away from the metaphors that underlie the prevalent culture's mindset towards those of the culture you wish to replace it.

Constructed Mental Image

When we get caught up in the excitement of a metaphor we often want to covert other people to our great revelations. We ramble away in meetings, we make powerpoint presentations, and we write articles and methodology books filled with our deep insights. Invariably, these fall flatter than we had hoped.

Metaphors are not that useful when they and their elaborations are written down, but when they reside in your mind, absorbed as rich mental models from which further insights can spring. Psychologists has discovered that the most effective metaphors are likely quite sparse in terms of words, but they have strong imagery value.

> I sat in a presentation where the author said that “agile software development 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”. It went on and on, and I certainly noticed a few eyes glazing over in the audience. It was overly wordy, and very dull. Far better, in my opinion would have been a slide of just three words with much higher mental imagery: “continual customer satisfaction”, followed by a long pause allowing the audience to think about its implications themselves, before the speaker continued.

High mental imagery, then, is vital for metaphors to work quickly and evocatively.

> Psychologists have demonstrated repeatedly that low imagery metaphors can only be retrieved by the mind slowly and serially. It is 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 it was learned. Repeating the alphabet backwards is much harder. Psychologists have shown that 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.

Importantly, though, effectiveness of a given mental imagery is very personal. There is no point forcing your own mental image of a metaphor onto somebody else for whom it is less evocative.

This might explain why metaphor experts have found that, perhaps surprisingly, drawings do not help communicate and teach metaphors to others. It seems that the process of drawing (or sketching) is itself a learning experience and without going through that experience yourself, any drawings you see can actually do more harm than good. There is research evidence suggesting that shared drawings are often counter-productive and lead people to interpretations you never intended.

So, don't force your drawings on people unless they ask for them. A drawing that is highly evocative in its mental imagery for you, may fall flat on its face for somebody else. I have found that a good time to create drawings is when standing with somebody at a whiteboard, walking through metaphors together. Both of you can quickly capture things you have uncovered together, sketching, scribbling, erasing, and redrawing as you go, while the metaphors are transferred to where they really belong, which is in your heads, not on paper or a whiteboard.

In summary, 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.

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 diving into analysis and design without having first created 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.

Appendix A : The Science Culture

Introduction

Before Manufacturing, Design, and Service Cultures began to dominate, there was a Science Culture. This began in the 1950s, when computer programmers wore white lab coats, and were expected to use scientific principles to ensure precise and predictable results. It never gained a particularly strong foothold in industry, since, despite its early promise, software development turned out to be far less scientific in practice that people had hoped. To some extent, the Science Culture has remained fairly strong in some University Computer Science departments, although even this appears to be on the decline.

Dijkstra

Perhaps the strongest proponent of the Science Culture 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, as they reveal some useful insights into the motivation for, the assumptions underlying, and the rise and fall of the Science Culture for software development. Here I will quote from many of those papers, and use the names that Dijkstra used for them himself (e.g. EWD268).

1950s : The Seeds of an Idea

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.

1960s : The Revelation

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)

1970s : The Proof, The Whole Proof, and Nothing But The Proof

Dijkstra then dedicated the remainder of his life to developing and evangelizing a Scientific Culture which embraced 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 mathematical proof should actually precede the program, forming its specification.

Dijkstra actually argued against the use of 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, software development is something very different, and needs to be rethought in its own right.

Dijkstra was certain that, despite popular opinion, software development is most definitely 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” (see EWD316)

Rather, “It is an intrinsic duty of everyone who professes to compose algorithms to supply a proof” (see again EWD316) The main activity of software development, then, is to create mathematical proofs that the programs 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 see it as a metaphor deeply ingrained in the Science Culture of which he was a part. 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 was inclined to interpret their own metaphors 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.

1970s and 1980s : Damn Those Amateurs!

Dijkstra's explanation for the early failures of the Science Culture was not that the programmers in white lab coats were trying too hard to be scientific, but that they hadn't been scientific enough! “Part of the blame should be put on the early scientists that got involved in computing ... The vast majority of them did not transfer their scientific quality standards to the programming that became their major occupation ... A generation of “scientific” machine users has approached the programming task more as solving a puzzle posed by the machine's manufacturer than as an activity worthy of the technique of scientific thought.” (see EWD924)

Throughout the 1970s and 1980s, Dijkstra argued repeatedly that creativity, innovation, and shrewdness had no place in software development. “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 ensure the computer industry transformed itself from craft to science.

To Dijkstra's increasing dismay, however, 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". Diagrams are, he claimed "for the uneducated and by the uneducated" (see EWD696). "Society needs competence and asks for quackery" he lamented (see EWD966).

By mid 1980s he had begun to believe 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, were 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" (see 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" (see 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)

1990s: Losing Hope

By the early 1990s Dijkstra appears to have almost given up hope of converting people in industry, to a Science Culture. He blamed this on 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 science. “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 computing science” (see EWD1284)

Requiescat In Pace.