The Pragmatic Programmers

Pragmatic exPress

The Stress Equation

Reduce Burnout,
Increase Happiness and Productivity

Marcus Lagré
edited by Julia Watson

Early Praise for *The Stress Equation*

I’m sure that I am not the only one that will recognize many of the problems that Marcus describes in the book and all of us would love some way to navigate the structures, context, and ways of working to reduce the stress. Marcus has given us such a tool and I can only hope that it gets picked up in many places—I know I will keep the Stress Equation handy where I go.

➤ **Marcus Hammarberg**
Lean & Agile Coach, Author of *Kanban in Action* and *Salvation: The Bungsu Story*, Scling AB

The Stress Equation is a thought-provoking and valuable read for anyone working in a fast-changing and complex environment. While software development is at its core, the generic principles and systems thinking approach are applicable far beyond. Recommended reading for individual contributors and leaders who aim for long-term high performance and well-being at once.

➤ **Rasmus Lind**
Founder & CEO, Learning Leaders Sweden AB

We've left this page blank to make the page numbers the same in the electronic and paper books.

We tried just leaving it out, but then people wrote us to ask about the missing pages.

Anyway, Eddy the Gerbil wanted to say "hello."

The Stress Equation

Reduce Burnout, Increase Happiness and Productivity

Marcus Lagré

The Pragmatic Bookshelf

Dallas, Texas

The team that produced this book includes:

Publisher:	Dave Thomas
COO:	Janet Furlow
Executive Editor:	Susannah Davidson
Development Editor:	Julia Watson
Copy Editor:	Karen Galle
Layout:	Gilson Graphics

ISBN-13: 979-8-88865-101-8
Encoded using recycled binary digits.
Book version: P1.0—August 2024

Contents

Acknowledgments vii

Preface ix

1. **Talking About Stress** 1
   - View Stress as an Occupational Health Risk 2
   - Define "Stress" 4
   - Discuss Stress on a Systemic Level 6
   - Wrap Up 9

2. **Team Cumulus: Developing Dilemmas** 11
   - Meet the Team 11
   - Increase of Pressure 12

3. **Identifying Sources of Pressure** 15
   - Create a Sense of Urgency 16
   - Structure the Work 19
   - Motivate the Individual 30
   - Wrap Up 38

4. **Team Cumulus: Building to a Break** 41
   - Team Responsibility 41
   - Increase of Complexity 42

5. **Identifying Sources of Complexity** 45
   - Understand Your Complexity Debt 46

Reduce Cognitive Load 49
Address System Inertia 59
Wrap Up 68

6. **Team Cumulus: From Clutter to Clarity** 71
Focus on Flow 71
Plan for Mutual Understanding 72

7. **Increasing Security** 75
Define "Culture" 75
Avoid Decreasing Security 77
Enable Increase of Security 83
Wrap Up 91

8. **Using the Stress Equation** 93
Discuss in Smaller Groups 94
Reconvene and Gather Thoughts 94
Collaborate in Retrospectives 95
Wrap Up 95

9. **Collaborating on Product Discovery** 97
Make It Visual 97
Manage the Portfolio 98
Discover the Way Forward 100
Get It Developed 102
Wrap Up 104

10. **Increasing Performance, Not Pressure** 105

Bibliography 107

Acknowledgments

This book would not have been possible without the support (and patience) of my wonderful wife Sara. Without the happiness and the numerous cups of coffee that she brings me, I would not have had the energy needed to see this project through!

I would also like to send a whole basket of gratitude to my partners at Snowdrop. You provided the positive and creative vibe needed when I ran out of steam!

A special shout out goes to my technical editor, Julia: you always said the right things at the right time.

Susannah, Tammy, and Dave. You provided excellent heavyweight input and feedback along the way. Thank you!

Cheers to Markus Klein, Theres Mangs, Marcus Hammarberg, Stefan Furenbäck, Kevin Gisi, Rasmus Lind, and Jared Duggan for taking the time to read the whole book and provide some much needed external reviews and feedback.

Finally, an extra special thank you to team Cumulus, the best team I ever worked with! Especially Anders, for showing me that successful leadership really can be a mixture of empathy and kindness, as well as determination and high expectations. (And a not insignificant amount of homebrewed beer!)

Preface

Welcome to *The Stress Equation*!

If you're holding this book in your hands, I'm guessing you're interested in improving your workplace. Well, you've come to the right place!

The Stress Equation model is a tool for discussing and analyzing the cause of stress in teams and organizations. By addressing stress as a systemic problem and moving focus away from the individual, the Stress Equation lowers the threshold for frank and open discussions—even providing topics and questions for group conversations on the subject.

Just to be clear, this book is not a guidebook on stress itself. Although it will be a central subject, I leave it to doctors and scientists to cover the depths and mechanics of stress as an organism's response to threats or challenges. I am, after all, merely a nerd who likes to improve things.

This book is based on observations made during my 20 years in software development. I have worked at many organizational levels, starting my career as a tester of mobile phone platforms, moving on to various programming jobs, and later ending up in management and coaching—predominantly as a "full-stack" Agile Coach. I have worked with everything from one-team Scrum to LeSS Huge organizations with 50+ teams.

I dare claim that my experience provides a rare holistic perspective of software companies.

Over the years, I've developed a special interest in stress and sustainable pace, using the reduction of stress levels as a key factor for continuous improvement. Why stress? Well, the simple answer is that I grew tired of watching people burn out due to organizational flaws beyond their control. I also grew tired of people not reaching their full potential out of fear of overwhelming pressure and complexity caused by disorganization.

With this book, I want to help people and companies better understand the causes and effects of excessive, harmful stress in their work environments and the factors I have observed to be in play in creating stressful situations. The book aims to help you explore the problem scope, not to provide an easy-to-follow formula to solve your problems. This is because every context is unique. No universal recipe exists—if it did, we wouldn't have any problems!

Who Is This Book For?

While stress is a widespread problem, and pretty much everyone is affected by it at one point or another during their career, I *would* like to say that this book is for *all the people working in software!* However, that's not the type of categorization that marketing departments like to run with. This is completely understandable since it makes their job nearly impossible. Sooo...

The book is aimed at team leads, Scrum Masters, managers, and other leadership roles. Since it's meant as a tool for continuous improvement, it's important that management is onboard and willing to commit. Otherwise, improvement is likely to stagnate, never spreading beyond limited portions of an organization. You'll learn that the overall organizational complexity, something that is typically outside a single team's or department's sphere of influence, contributes to creating stressful situations. Changing these situations requires leadership commitment and alignment on a higher level.

With that said, if you feel you are suffering from stress but are not in a managerial position, maybe this book can give some insights into why you are feeling stressed and help you find ways to address it by promoting positive change.

Though the concepts are applicable in many organizations, the examples provided are primarily targeted at software companies since that's where my experience lies. If someone outside of software reads this book and finds it useful, don't hesitate to get in touch. I would love to hear your take on it! I am pretty easy to find on LinkedIn.[1]

Given that I have a background as a Scrum Master and Agile Coach, the book is written from that perspective. I will assume the reader has some familiarity with Agile lingo, but I doubt you need to be an expert to absorb the concepts I'll try to convey.

1. https://www.linkedin.com/in/marcus-lagr%C3%A9/

How to Read This Book

While the different chapters and segments can be read separately, providing bite-size insights into particular areas, I suggest reading the book from start to finish to understand the concept behind the Stress Equation.

That said, I am a big fan of book clubs, and each chapter provides ample fodder for thought and discussion—even providing questions to discuss on each topic.

These can be used to build improvement workshops, provide inspiration for retrospectives, or to increase the overall understanding of the stress you might be feeling in regards to your work environment.

In other words, you can approach the book in whichever way suits you, but I have intentionally kept things brief to make it a quick read.

What's in This Book

The Stress Equation is visualized as a mathematical equation to explain how different sources of stress interact—the variables being *Pressure*, *Complexity*, and *Security*. Each variable has a chapter of its own.

Pressure deals with the different sources of urgency we may encounter, such as the influx of work items or how we set deadlines.

Complexity will explain how the way we build our products and organize ourselves can reduce our ability to finish work.

Security addresses the cultural aspects of our work environment and how a sense of security is vital to lowering the overall stress levels within teams and organizations.

Interspersed with these chapters will be a story—or case study—of a team I worked with. I'll detail how we found ourselves under increasing pressure and complexity, and how conflicts affected our security. There will also be short anecdotes and insights that I have gathered over the years, relating to the topic at hand.

Finally, there will be a wrap-up and some hands-on examples of how to use the book in your improvement work.

Stress is too big a problem in our work lives to be solved on an individual level. It's about time we talk about it as a systemic problem, and the Stress Equation is a helpful tool to get you started.

There's no reason to wait: let's talk about stress!

CHAPTER 1

Talking About Stress

How well do you cope with stress?

I have lost count of the number of times I have been asked that question over the years in interviews for jobs or consultancy work.

It's a strange question because what kind of answer can one expect? Who in their right mind would out themselves as neurotic when trying to impress the employer with their impeccable skills?

I have never had the guts to turn that question around, though that would have been an interesting experiment.

What flaws in your organization, planning, and leadership necessitate you to ask that question?

It's a reasonable question to ask because stress is costing lives. Not necessarily in a literal sense, even though stress is a proven contributing factor to a vast array of mental and physical ailments.[1] No, when I say it's costing lives, I'm talking about how stress can drain the joy of life from a person. It can cause negativity and strain that spills over from work life onto private life and stops you from having the energy to do more than eat, sleep, and work.

In Sweden, where I live, stress is the leading cause of mental illness and sick leave among academics and knowledge workers and has been for quite some time.[2] Looking at Gallup's "State of the Global Workplace 2023" report,[3] employees have been reporting an increased feeling of stress over the past decade and a half.

1. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5579396/
2. https://www.saco.se/globalassets/start/opinion--fakta/rapporter/2021/2021_psykiskt-ohalsa-bland-akademiker.pdf
3. https://www.gallup.com/workplace/349484/state-of-the-global-workplace.aspx

This would suggest that stress as an occupational health hazard isn't a local problem for the Swedish workforce but a global trend.

View Stress as an Occupational Health Risk

What sets stress apart from other areas of occupational health is the focus on individual susceptibility and coping strategies—something pointed out as early as 1985 by Dean B. Baker in his article, "The Study of Stress at Work."[4]

In most other areas of occupational health, there are laws, regulations, and recommendations aimed at addressing hazardous work conditions to minimize risk to employee health and safety. This responsibility lies with the employer.

For office workers in Sweden, this might be applied as ergonomic recommendations and policies on the minimum requirements for an acceptable workstation. For example, it could detail what a good office chair and desk should look like or how often we should stand up during the workday so we don't bust our backs. We have appointed safety representatives to check that there are defibrillators on every floor in the event of a cardiac arrest, and there's at least one fire drill every year.

Even though Sweden has specific regulations aimed at addressing the psychosocial aspects of work environments,[5] which include stress and reasonable workloads, the regulations seem hard to apply in practice, as stress remains a major cause of sick leave.

It appears that companies have a much harder time being proactive with psychosocial health risks than physical health risks. Companies that I have worked for over the years have offered psychological counseling as part of their health benefits package, but it has almost always been a reactive measure, not kicking in until after a person has gone over the edge into burnout. It's a little like offering a hard hat to a construction worker *after* they get hit on the head.

Once again, even if offering counseling is a great practice, it focuses on the individual's responsibility for coping and handling stress. It doesn't focus on fixing the work environment, only the personal perception of it.

I worked for a company struggling with high burnout numbers within management that "solved" the problem by introducing personality tests and subsequently only hired people who scored very high on stress tolerance. Since

4. https://www.annualreviews.org/doi/abs/10.1146/annurev.pu.06.050185.002055?journalCode=publhealth
5. https://www.av.se/en/work-environment-work-and-inspections/publications/provisions/organisatorisk-och-social-arbetsmiljo-afs-20154-foreskrifter/

this was a software company, not an army company, where people get sent into combat, I found this to be a questionable practice. Surely there must be something wrong within the organization if your hiring practices signal that stress tolerance is the most important management skill in software?[6]

Such a hiring practice could potentially create a personality monoculture within management, with even less understanding of stress-related issues among the employees.

Coping with Burnout

I am myself a sufferer of stress. My own story was caused by both private and work-related issues. When my kids were little, they didn't sleep. I know this is a common story, but my wife and I were suffering badly. For six years, both children woke us up every night and needed physical closeness to wind down and go back to sleep. Years of disrupted sleep messed up my internal body clock and triggered chronic insomnia.

While that was taking place, my work became increasingly demanding. I was a programmer in a team buckling under technical debt and growing conflict within management, which spilled over onto us. My inability to perform at work caused a stress loop that, in turn, worsened my insomnia.

Eventually, on a Saturday morning, I found myself unable to get out of bed. I could not for the life of me get my legs over the side and get up. For about two weeks, I could barely eat and only managed to leave bed to go to the bathroom. Nature's call is a strong motivator!

I was on sick leave for two months, spending my time with light exercise, meditation, and rest. My wife pulled a heavy load, and I am forever grateful for that. Normally, we split all household work and responsibilities 50/50. For weeks and months, she did it all while working full time, which gave me a whole new sense of admiration for single parents. She even slept on a mattress outside our bedroom door to physically stop the kids from entering at night.

Slowly, I came back to life, occasionally even sleeping a full eight hours.

Though I did make a full recovery in the end, burnout comes with scar tissue. Your stress tolerance is never quite the same again. I can only compare it to a torn ligament in the knee. Even though the knee heals, you simply can't put as much weight on it as you used to, so you are constantly aware of how much strain you are putting on it. It becomes something you have to be forever mindful of.

6. https://hbr.org/2017/04/employee-burnout-is-a-problem-with-the-company-not-the-person

I can't fault my employer for my kids waking up every single night, but there were mechanisms at work triggering more stress. Through my work as an Agile Coach and my openness about my stress history, I have ended up in a lot of conversations and situations where I have collected insights into common stress triggers in software development organizations.

I will try to summarize these insights throughout this book. While the examples will be oriented toward software development, I am sure most of them apply to other fields of work as well. One thing that makes software development special, though, is the potential level of complexity and how fast it can become a source of stress and frustration if not managed properly or taken into consideration when planning.

Before moving on, I think this would be a good time to define stress so we're on the same page about what we're talking about.

Define "Stress"

Almost all definitions of stress—not counting anything relating to materials science or emphasis on important details—are focused on the individual. Stress is ultimately a subjective sensation.

However, when talking to teams about stress, I often find the perception of what stress is, how it feels, and where it comes from to be very narrow. Stress is not always working long hours or a jam-packed work calendar. There is more to it than that—more parameters to consider.

When I talk about stress, I use the following definition because I think it catches the multifaceted nature of stress:

> *Mental or emotional strain as a result of adverse or demanding circumstances.*

Mental *or* emotional strain.

In my experience, emotional strain in software development rarely comes from the work itself but rather from having to work in a particularly toxic environment. I have personal experience with toxic environments and know the toll it takes. When emotional strain hits you, it is very in-your-face and almost impossible to ignore, as our entire being forces us to listen. My point is that emotional strain is often noticeable.

The *mental* strain, on the other hand, is a little trickier to catch, and from my perspective, it's a lot more common in the software profession. As academics, knowledge workers, or engineers, we are spurred by mental challenges, such as solving puzzles and analyzing problems. We are probably not always well equipped to recognize when a mental challenge transitions into mental strain—at least, that’s my conclusion.

This is especially true for the ambitious among us, who are used to applying ourselves even harder when meeting resistance. The solution is probably just one more push away, so we continue, thinking that if we keep at it just a while longer, we’ll crack it. I have seen many people fall into that trap, including myself.

Similarly, it’s easier to identify a demanding circumstance than an adverse one because the former is more in-your-face.

A demanding circumstance is pretty obvious. There might have been a short deadline, so we had to pull together and plow through work all weekend, but in the end, we made it! We recognize them and hopefully understand that we can’t have pressing deadlines that necessitate overtime every week or every month. We need recuperation before the next demanding circumstance, or we will wear ourselves out.

Adverse or unfavorable circumstances, however, are harder to pinpoint because there is usually no single source. Adverse circumstances are often the result of a cocktail effect of several small inefficiencies, annoyances, or conflicts that we have to maneuver around.

On their own, each of them is no big issue, but the sum of them adds up to one large energy vampire or time trap. And they rarely go away on their own. They’re always there unless we actively address and fix them.

Coping by Quietly Quitting

After the Covid-19 pandemic, *quiet quitting* was the buzzword. Quiet quitting has probably always existed in one form or another. Remember that German guy who, when retiring in 2012, admitted he had done absolutely nothing for the last 14 years of his employment?[7]

So while the phenomenon is nothing new,[8] the post-Covid debate put the spotlight on a very neglected aspect of work life: the realization that circumstances in the workplace are too demanding or too adverse to allow for any kind of work satisfaction, so you might as well just put in the bare minimum.

Quiet quitting becomes a defense mechanism. The result is a growing indifference because caring about your performance and feeling responsible for

7. https://www.thelocal.de/20120411/41879
8. https://hbr.org/2022/08/quiet-quitting-is-about-bad-bosses-not-bad-employees

the results would likely cause you to spontaneously self-combust into a fireball of frustration. This is such a loss of potential joy because personally, I think taking pride in your work is one of life's major sources of happiness.

Don't you think it's time to do something about that trend? To become aware of where these adverse circumstances emerge and which unnecessary stresses we are struggling with?

We need a way to talk about stress that does not put individuals in focus. We need a way to put the focus on the system and circumstances that cause the stress. This is the reason I created the Stress Equation—as a model on which to base discussions and analysis and to find the sources of stress and frustration on a systemic level.

Because if we can't talk about it, we can't fix it.

Discuss Stress on a Systemic Level

The idea for the equation came about when I was suffering from burnout and needed to analyze my stress reactions to understand why this happened.

I had been working as a developer on a team called Cumulus for a couple of years on a world-leading product in logistics optimization and forecasting. We were, however, facing mounting problems related to aging technology and a challenging system architecture that had been growing "organically" for decades.

These challenges hampered our ability to deliver efficiently and limited what new functionality we could realistically add to the product. This, in turn, led to increasingly strained relations between software development and product management that, over time, would deteriorate into outright conflict and open hostility.

Defining the Stressors

During my recovery from burnout, I ended up dividing my stressors into three separate categories:

- Having a lot of things to get done
- Dealing with complex problems or contexts
- Having to deal with conflict

I concluded that I could easily handle having a lot of things to get done and dealing with complex problems, but preferably not at the same time. Having to deal with conflict was, however, the most critical one for me. Constant and ongoing conflict sucks me dry of energy very quickly.

After many discussions on the subject with colleagues and friends, it seemed that most people could relate to these categorizations. Working under pressure with complex problems in an insecure work environment seemed to be a combination that would tip most people over the edge.

While working as an Agile Coach in several organizations, I kept making mental notes of what appeared to affect the stress levels of the teams I was working with. I also noticed a reluctance to talk about stress in some teams, especially if the psychological safety of the group was on the lower end—when team members were uncomfortable with being completely open about personal weaknesses or shortcomings.

So, not only would a model that focused on the systemic causes of stress be helpful in finding the root causes, but it would hopefully help people to open up by removing focus from them personally.

The idea that I ended up with was the Stress Equation, visualized as a mathematical equation in the following illustration.

$$\boxed{\frac{\text{PRESSURE} \times \text{COMPLEXITY}}{\text{SECURITY}}}$$

The equation has three variables representing the different categories that I had identified as my own stressors.

- Pressure (having a lot of things to get done)
- Complexity (dealing with complex problems or contexts)
- Security (having to deal with conflict)

This makes the stress level equal to pressure multiplied by complexity divided by security.

In other words, the stress level always increases with the pressure under which we operate. Likewise, the stress level increases with the complexity of the problem we are trying to solve or the complexity of the context in which we find ourselves. Given the multiplication, the stress level skyrockets when there is high pressure in combination with high complexity.

However, the stress level always decreases with a high sense of security.

Now, just to be extra clear, I did not set out to create a scientific model to quantify stress. I needed a practical way to capture different aspects of work stress, help categorize and analyze different sources of it, and lower the bar for discussing the issue in the teams I was working with.

The equation will not allow you to mathematically quantify your organizational stress level; it's a tool for thinking and talking.

Don't Put Numbers into It

The equation is not meant to be used as a literal mathematical function. It's meant to be a model for thought and visualization. Engineers, please don't try to put numbers into it!

Don't Strive for Zero Stress

The stress level in an organization can—or at least should—never be zero. Without any pressure, there is no sense of urgency, and performance will drop. Without any complexity, we have to ask ourselves why this work wasn't automated a long time ago. And with infinite security, we would probably be sitting in the sun sipping colorful drinks, getting no work done at all. I know I would.

So, this is not about minimizing the numerators and maximizing the denominator but rather finding a balance between them to create a sustainable and efficient operation.

To "speak management," one might say:

- Tactical decisions affect Pressure
- Strategic decisions affect Complexity
- Culture affects Security

What do I mean by that?

Tactical decisions determine how much work we feel we need to do and how much time we have to do it. Strategic decisions decide our direction in the long term—the direction that our tactical decisions should take us over time.

Company culture decides our sense of security. A company can, for example, have generous pay and benefits but a culture where snide remarks are tolerated or where "feedback" from upper management comes swiftly and hard. On paper we are secure, but in reality, we're out in the cold.

Tactical decisions can have a quick impact. If we are willing to make some tough prioritization and start paying attention to how we structure our work, the pressure parameter can be affected very quickly.

Strategic decisions take longer to take effect. If we are struggling with quality issues or bad product design that increases our complexity, this requires a lot of work to overcome and manage.

Changing the culture of an organization to build trust and psychological safety takes a significant amount of time and effort. Building a culture of collaboration and transparency can take years.

I would like to point out that the examples I give for each variable are not an exhaustive list of everything that might be in play. There may, for example, be other sources of complexity in your context. The examples I give merely summarize those I have experienced and identified myself. The model is there to help you analyze and categorize your own context, so don't be afraid to add things I haven't mentioned.

See the examples as a guide, not a blueprint.

Wrap Up

The Stress Equation was created as a tool to shift focus away from individual coping strategies and onto systemic issues and company strategies. Stress is too big a problem for us to solve by addressing our individual shortcomings; we need to look at the organizational root cause and solve it there.

The Stress Equation has three variables that interact to describe the causes of stress:

- Pressure (having a lot of things to get done)
- Complexity (dealing with complex problems or contexts)
- Security (having to deal with conflict)

We need to understand all of these variables to get the full picture of stress from an organizational perspective.

But before we move on to the first variable of the equation, Pressure, let's get acquainted with team Cumulus and their situation when I joined them.

CHAPTER 2

Team Cumulus: Developing Dilemmas

Cumulus was the software team in one of the company's product departments. The product was, in short, a software suite for follow-up, forecasting, and optimization of logistics costs. The software was accessible to customers via a web app, but professional services were also sold on a consultancy basis, where a team of experts and support personnel provided deeper analysis and real-time advice on a request or subscription basis.

Meet the Team

The team set-up was inspired by the Scrum playbook—except we had no dedicated testers. The development manager acted as Scrum Master, and a designated domain specialist acted as product owner, but we didn't use those titles because we didn't do full-on Scrum. The team's work was more of a DevOps nature, with a heavy lean towards Ops. We had too much unplannable maintenance work requiring hasty attention for proper sprint planning to be meaningful. That's a common situation in DevOps contexts, so we worked according to Kanban principles but kept a monthly cycle of retrospectives and planning.

Since part of the product and services was based on monitoring costs, we constantly received large amounts of data for analysis. Our experts had access to more tools and functions than we offered externally via the web app, so our in-house service was always better than what the customers could produce. Let's say it was a way of making sure they kept calling for premium service.

All in all, we had everything in place to provide a world-class product and service—and it *was* a world-class service. We had less than a handful of credible competitors in a global market.

But, as I have found out from talking to other developers with experience working on world-leading products, what you see up front and what's going on in the back does not always match. A polished front end is sometimes held up by nothing else than service-minded developers and loads of gaffer tape (aka batch scripts).

Increase of Pressure

The product and services had been continuously developed for more than two decades, and the legacy systems were becoming an increasing cause of problems. Over the years, developers had come and gone, with consultants coming in periodically to solve specific issues, which they did but not always with consideration to the overall tech fauna or high load resilience. Before our development manager joined, a year or two before I did, I was not entirely sure who was in command of the software parts. From the looks of it, no one.

The entire product was a patchwork of software systems—scripts, desktop applications, REST APIs, cron jobs, FTPs, web interfaces, a huge monolith of a Silverlight monster. You name it, we had it!

It was complex, most of it lacked proper documentation or unit tests, and integration testing was almost nonexistent. Sometimes, when pushing to production, there was no telling what kind of butterfly effects might be set in motion. Changing a parameter at one end could cause exceptions in another, which might not be noticed until days later. It was a fairly unstable system, and many subsystems were in bad shape or aging poorly.

Most of our work was simply keeping things from breaking or fixing things we'd broken when fixing something else.

Even so, I can say with confidence that there were few—if any—forecasting algorithms as comprehensive as ours. The guys who had developed the algorithms had 40+ years of experience each and had refined the calculations for decades.

This was part of our predicament. These guys were closing in on retirement and were domain experts, not educated programmers. The algorithm code lacked sensible structure and was written in Fortran 90. To keep developing it, we needed to rewrite it with more structure in a modern programming language. We also wanted to get rid of the numerous sections with comments proclaiming:

```
!Here be dragons
```

Even more pressing was the Silverlight issue. Microsoft had set an end-of-life date for the technology, and browser support was already dwindling. That application was, by and large, what we were selling to external users, and it needed to be replaced in its entirety.

There had been a strained relationship between the development team and the product manager for quite some time. He was not very willing to pay (a lot!) to replace functionality he considered he had already paid for. This is not an uncommon situation. Non-software people (understandably) have a hard time grasping problems surrounding aging systems. He was also convinced that software engineers would gold plate everything if not kept on a tight leash—which, let's be honest, is true—so he'd push for us to release things, scolding us for gold-plating the software when all we were trying to do was add some rust protection.

This was more than a rust problem, though. Our entire product was about to become deprecated! At the same time, he was frustrated at the pace of our delivery, and when we tried to explain the technical challenges we were facing, he treated them as cheap excuses. To our frustration in return, he had a habit of making promises to customers without consulting us, constantly underestimating the effort and time needed. He also kept insisting we needed to bring on more customers and add new features to our software, while we insisted on modernizing the system and replacing Silverlight as soon as possible.

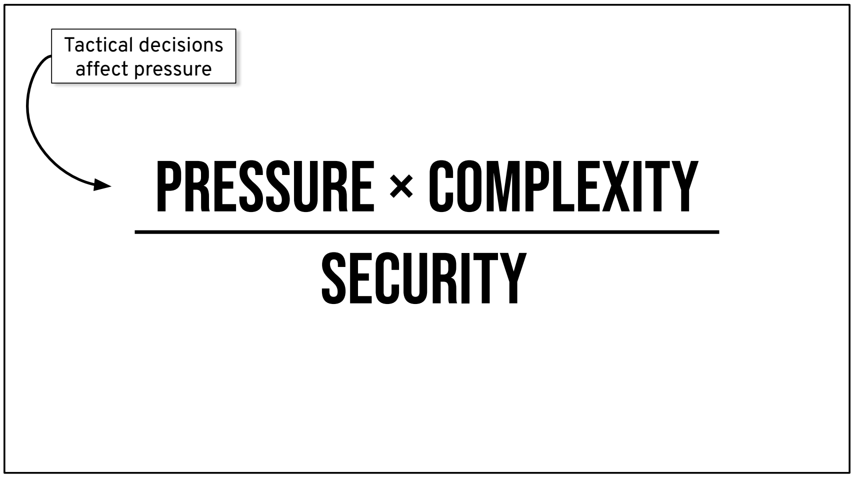
The team was facing a high-pressure situation. An increased load on an already unstable system left us with even less control of our influx of work. Furthermore, the team's high sense of responsibility was biting us in the backside. We knew what had to be done but were not given the go-ahead to do it. With a looming end-of-life on Silverlight and knowing the amount of effort needed to replace it, our sense of urgency increased with every passing week.

CHAPTER 3

Identifying Sources of Pressure

Pressure is affected by our tactical decisions, and by tactical decisions I mean how we decide what we should work on—how much work we have in our backlog, and how much time we have to get it done. In other words, it's defined by how we prioritize, set deadlines, or decide when something is finished.

The following image illustrates how tactical decisions relate to the Pressure variable of the Stress Equation.



In this chapter, we will learn about sources of pressure and what we can do to address them. There will be questions provided to either ponder on your own or discuss in your team. Our goal is not to remove all sources of pressure, but rather to identify the ones that are unnecessary or unconstructive. By doing so, we can focus on what gives us a productive sense of urgency and a feeling of "working on the right things."

There are a number of ways of doing this, but without understanding the context in which a method is being applied, there is no way of telling if it will work. So, while I could play the snake oil salesman, providing you with a tonic for every ailment, I suggest you explore the issues you may be facing by asking the questions provided throughout the chapters. Only when we properly understand the problem can we take the correct approach to solving it.

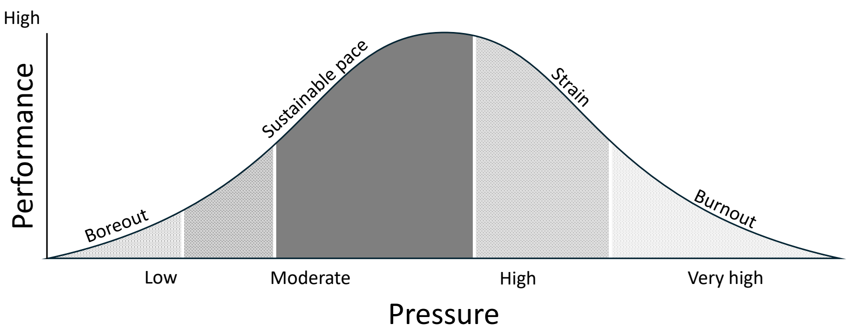
What I can tell you for sure is that without a method of regulating pressure, there is a tendency to gravitate towards the extremes, where you either find yourself drinking from a firehose or perpetually procrastinating.

Create a Sense of Urgency

Though a source of stress, pressure in and of itself is not bad or evil. Without any kind of pressure or sense of urgency, we would probably not be very productive. In fact, most of us find working under pressure stimulating, as long as it's not an overwhelming amount or over too long a period of time.

Courses on leadership and motivation normally have at least one slide containing the distribution curve illustrating how pressure regulates performance.

The following image illustrates the Pressure-Performance curve, with performance on the y-axis and pressure on the x-axis.



This graph highlights that low pressure will generate low performance. This is when you feel indifferent toward your job due to a lack of stimulation from too few or too simple tasks. With too little to do, the sense of urgency drops, and we find ourselves procrastinating. It may eventually lead to a kind of inverted burnout, sometimes referred to as "boreout," where you get depressed from under-stimulation.

At the other end of the x-axis, we have very high pressure. With too much pressure, performance drops either due to mental overload and potential burnout or indifference and resignation caused by unrealistic commitments. This is when the backlog feels like digging a hole at the beach, with the waves constantly filling the hole with sand again. We'll never finish digging, no matter how hard we work.

The idea behind this graph is to illustrate how studies show that there is an optimum level of pressure that will generate a maximum performance.[1] You want to stay on the left side of the curve because that is a sustainable situation. Once we've passed the peak, not only does performance drop, but the quality of work does as well.[2] We are also in a straining situation that will likely be unhealthy in the long run.

While this is a good illustration to drive the point home, I have yet to work with a company that actually measures the pressure it's under in regard to its own performance. As someone who likes data, this is a little bit of a surprise. Companies and departments working with software normally have some kind of ticketing system, keeping track of the number of change requests, errors and bugs. This should provide ample data for pressure and performance. In an Agile or Kanban setting this might be User Stories, or even sticky notes on a whiteboard—whichever way you prefer to visualize your workflow. This data is rarely treated as an important source of input for improvements.

I usually describe the sense of urgency we feel as the relationship between the amount of work we perceive that we have to do and how much time we have to complete it. One stressor for development departments is when they lack the structures needed to make the strategically correct tactical decisions on backlog handling, prioritization and refinement. If we lack this ability, we cannot decide what's important and will likely end up in one of the aforementioned extremes—drinking from a firehose or procrastinating.

This structuring and refinement is often neglected, especially when pressed for time. Descriptions of work items become fuzzy one-liners, and the backlog is treated like the simplest of data structures: a stack that you push and pop things to and from. Last in, first served.

Enable Clarity with Structure

Some organizations that adopt Agile as a more lightweight requirements process to get to development faster haven't always realized that part of the problem to begin with was a lack of competence in specifying goals, priorities,

1. https://onlinelibrary.wiley.com/doi/10.1002/cne.920180503
2. https://ntrs.nasa.gov/api/citations/20060017835/downloads/20060017835.pdf

and user needs. This is a whole discipline in itself, but it's rarely treated like one. There are all these jokes about how users and customers are unable to express what they really need. Well, we are not always that great at helping them—and, by extension, helping ourselves—either.

Historically, companies have tried to bridge the competence gap in requirements collecting with processes, templates, lists, charts, and documents. This is all done in an attempt to improve how to express what they want, but it only ends up complicating things by communicating via documentation rather than face-to-face. At least, that's my take on why we ended up with such cumbersome and bureaucratic project frameworks.

Face-to-face communication is absolutely the preferred method, but it doesn't scale well for most organizations. If you have a simple product and one dedicated development team, set up exactly like the Scrum Guide[3] describes, this mode of operation works. This is, however, not the reality for the vast majority of teams.

There is rarely time and opportunity to meet users or stakeholders on a daily or weekly basis. Even when there is, without knowing how to translate the conversations into effective goals, we will likely miss the target anyway. Communication can easily become inefficient and frustrating.

I meet many software departments who are stressed out because they lack a disciplined way of breaking down larger work packages and structuring backlogs. There is little or no overview of status, progress, and priorities. Ideas are put straight into development without a controlled maturation process or refinement.

Development on a backlog item quickly comes to a halt when the developers start asking questions that no one readily available can answer. For them, rapid face-to-face communication with end users is a utopian dream. Time with a product owner might also be strictly limited. But, they have learned that documentation and requirements are a bad thing, so they haven't spent any time defining, refining, and structuring the work.

The result is that the developers set the backlog item to "Blocked" while awaiting a response and pop a new item from the stack into "In Progress" so they don't sit idle—increasing the WIP in the process.

How we structure and clarify our work is an important aspect of creating a sense of urgency and focus.[4] Without an overview of status, progress, and

3. https://scrumguides.org/
4. https://www.crisp.se/bocker-och-produkter/prioritera-fokusera-leverera

priority, most teams either end up with no sense of urgency on anything or a sense of urgency on everything. Either way, focus is lost.

Enable Focus with Clarity

When there is no sense of urgency on anything, teams tend to get stuck with busy work, mostly endless bug fixing, even if they're non-urgent or low-impact bugs. If we had a good way of prioritizing, we would realize that these things are economically unsound to spend time on. This ends up causing stress for any project manager or stakeholder who is waiting for new development to get delivered.

On the other hand, if there's a sense of urgency in everything, there's usually stress in the team because we try to make progress by task switching, but nothing ever gets finished. There is never proper closure. Everything is always up in the air. We feel responsible for meeting the demand, which is commendable but rarely lays the foundation for a sustainable pace. High pressure of this type, over a long period of time, will cause us to start taking shortcuts, creating technical debt and increasing the complexity of our software.

Tactical decisions can have a quick impact, so when we are feeling stressed a good place to start is to get some proper structure and an overview of our work.

Questions About Pressure

- What creates a sense of urgency for us to get things done?
- How can we create a balance between performance and sustainable pace?
- What creates unnecessary pressure on our team/organization?
- Is it easy for everyone to understand what our most important, highest-priority tasks are?

Structure the Work

We don't talk about Work Breakdown Structures (WBSs) the way we used to—particularly in the Agile community. WBSs are ways to visualize work by breaking it down into hierarchical trees. Perhaps we bundle them with the memories of UML schemas, GANTT charts, artifact lists, and other fun documents we used to do in proper Waterfall projects. You know, way back when we wrote the code three times in different charts and specs before we actually wrote the code—and then realized it wouldn't work.

Those were the days! (I am, of course, kidding.)

However, visualization and structuring of work are often neglected art forms, something we would do well to spend more time on in the Agile world. There is a widespread idealistic tendency to view any and all requirement documentation as "waste." In fact, even calling them "requirements" can be a provocation because, in Agile, we work with user-centric Stories, not system-centric requirements.

Instead of detailed requirements, rapid feedback loops on finished code are the idealistic way of building software iteratively. While I do not object to the importance of rapid feedback cycles, in reality, cycles are not fast enough in most contexts for this to be the optimum way of working. In these cases, collecting, clarifying, and refining backlog items has to be part of the iterative process. In my way of seeing things, this is just another way of collecting and documenting requirements.

Tools and methods might vary, but the goal is always the same: clarification and structuring of what the heck we are supposed to be doing!

Don't Skimp on Preparations

When we focus too much on getting to the coding quickly, we forget to structure and define what we are supposed to do. It stops us from being effective, particularly with larger undertakings.

It's not uncommon to have initial resistance from developers to do the necessary refinement work. Suggesting that we go for another round of defining and breaking down a work package, maybe even formulating a rudimentary plan, will trigger some developers to point at you and scream "Waterfall!" in an *Invasion of the Body Snatchers* kind of manner.[5]

Okay, maybe that's a bit overdramatic—though only a little—but we shouldn't take the Agile Manifesto's value, *"Working software over comprehensive documentation,"* as saying "no documentation whatsoever." We need to remember this was a reaction to documentation-heavy, often very bureaucratic Waterfall processes that put too much emphasis on the importance of the plan.

While we shouldn't overcomplicate things, without proper planning and refinement, we risk scope creep or hitting landmines that could have been avoided. Worst case, we read the Agile Manifesto as saying: *"Barely working software over bothering with documentation."* An hour spent defining the work ahead can easily save days or weeks in lead time and help keep momentum up. We need an execution plan.

5. https://www.youtube.com/watch?v=wTP_SdjD5ms&t=112s

Without proper acceptance criteria and well-defined stories or tasks on how to reach our goal (which is what we should achieve with our refinement), there is no way of determining the status of development once in progress. Think of it as a travel plan (I deliberately avoid the term "road map" here). The acceptance criteria is the destination, perhaps collected in what some Agile organizations would refer to as an Epic and stories and tasks are the waypoints. It's an easy way to show where we are. If we get interrupted or have to stop work for some other reason, we quickly know where to pick it up again. We also need an idea of the waypoints to better understand the effort involved in the journey.

"But I have it all in my head," you might be thinking. Well, first of all, no, you don't. And I say that from observing others *and* myself. Second, that's beside the point. This breakdown creates a joint vision and understanding of where we are going.

Chances are there are a lot of steps we haven't thought of and obstacles that will become obvious just by sketching the travel plan out. Plus, if someone else wants to get a quick status overview, there is a reduced need to disturb you with questions about how things are going. We can simply look at the number of finished tasks and the rate at which we close them to figure out whether we appear to be on track or not.

We can call this pre-coding refinement a "No-Code Proof of Concept" if it helps us get past the resistance to do the work.

Plot the Travel Plan

When I act as a project manager or Scrum Master, and there is not even a travel plan—like a well-defined Epic containing a rough outline of stories and tasks—we are not going to start the journey. The trick is to keep it lightweight and never see the travel plan as set in stone. We respond to change by updating the plan as we go, but starting development without that plan will only get you barely working software, I assure you.

I know I sound like a bore, but this is part of the discipline we need to master to deliver reliably and efficiently. The easiest way to get a team to deliver more effectively is to be a backlog stickler and an enforcer of WIP limits. It's a basic thing in both Agile and Lean, but it is still something we get careless about. I am not sure why that tends to happen. Maybe it's because the reward of the work is delayed.[6]

6. https://www.researchgate.net/publication/22054716_Specious_Reward_A_Behavioral_Theory_of_Impulsiveness_and_Impulse_Control

In a small team, in a small context, communication is easy, and we can afford to be more relaxed about it. When we scale things up, either the size of the work or the size of the organization, we need to get more serious.

If we don't have a clear overview of the status of what's in progress, the risk of having little progress is higher. When a team has a habit of working for weeks on one-liner Jira tickets, how can we make informed decisions on when to inspect and adapt? How can we tell where we are or how far we have left to go?

Slow progress is often a source of stress and frustration, particularly in a traditional *you build it, you run it* setting, where development easily grinds to a halt when operational realities take up more and more time.

If we are constantly interrupted and need to switch to urgent maintenance tasks, for example, we lose track of where we are and have to waste both time and cognitive energy to find our way back again. Interruptions and lack of overview were challenges for Cumulus, which definitely impeded our ability to finish and deliver things efficiently.

In my experience, breaking work down into relatively fine steps right before development starts (using sub-tasks or checklists, depending on the ticketing system) is really important when there is a lot of involuntary context switching.

It is essential that we have well-defined tasks when we suffer from one of progress's main foes: the badly regulated influx of work items!

Control the Influx of Work Items

Every software team I have ever coached or worked in has had a higher influx of ideas, suggestions, and demands than we've had the capacity to finish. This is why it is so important to set the right priority on incoming work. If we jump on everything that comes at us, we risk working on the wrong things. Our sense of urgency is created by novelty rather than what generates maximum use and utility per invested development hour. As previously mentioned, this is when we start treating and using the backlog more like a stack than a properly prioritized list of work that should be taking us closer to a bigger goal.

Focus on Quality and Automation

From what I have seen, the number one reason teams get stressed out is that they constantly get interrupted by quality issues or manual tasks that could be automated or delegated. Bugs and recurring manual tasks are often viewed as operational work, which is given a fast lane into active development. We don't evaluate or question it; we just do it, which eats away at our time for adding new features—presuming we have a *you build it, you run it* approach.

This can be a vicious cycle because those quality issues are likely the result of shortcuts taken when under pressure to deliver on time. If we're not allowed to clean up and stabilize, we'll have constant pressure on the team caused by quality issues and production hiccups.

Similarly, lack of automation is predominantly a decision made during a high-pressure situation, when we think we will do automation later, but later never comes. Or at least, not until we've reached a breaking point when things can no longer wait, and our hand is forced. The good part is that we finally take the time to automate. The bad part is that we do not control our influx of work items.

Every time our hand is forced is a time we did not make the decision—circumstances determine our actions, not our planning. That lack of control is a common source of anxiety and something every team and software company should strive to be on top of. It can create further reluctance towards planning and backlog discipline because these are made meaningless due to constant, urgent interruptions.

Let Ideas Mature

The second most common reason team output suffers, with stress as a result, is bringing immature ideas to the backlog. This is normally the result of an inability to say "no" or to deprioritize work.

Product organizations lacking a proper product vision end up in varying degrees of serfdom under their customers. They are unable to say no to demands that may, in the long run, hurt the product or the company's profitability. We will talk more about this in the chapter on complexity, but regarding to the influx of work items, we need to have a way of saying "no" or "this needs clarification."

The best thing we can decide on to protect ourselves is a Definition of Ready. In other words, we need to define the minimum level of preparatory work we require of customers and stakeholders (and ourselves) before even considering putting something into active planning or development. This is a good sanity check of how important this work item really is. If the customer or stakeholder isn't prepared to invest time in refining and defining it, how important can it really be?

There is more instant gratification in saying yes than saying no, but unless we can regulate what we say yes to, we have no control over the influx of work. A Definition of Ready is a minimum standard to have in place to put more of the responsibility on customers and stakeholders before we accept work into our backlog.

Trim the Tail

The third source of stress for teams regarding the influx of work items is the inability to remove items that made it into the backlog long ago. If, for whatever reason, something gets stuck in backlog limbo, how do we address it? How old can an item get before it is eligible for deletion?

If we don't regularly trim the tail of our backlog, we will constantly send the message that our team never delivers fast enough and that we are always behind. This is the message we will send to our stakeholders and to ourselves. And unless we stop making promises we know we can't keep—stop keeping backlog items we will never get around to—this message will also be true.

Go Fish!

I was coaching some product owners whose teams were stressed out by the number of items in their backlogs. They felt they were behind and were disappointed that their output only made small dents in the numbers. Refinement and prioritization took forever due to the sheer amount of stuff to go through. Things were never deleted, only down-prioritized. There were four to five-year-old bug reports floating around, eating time and energy every refinement cycle. They weren't always discussed in detail, but the team was reminded that these items were still there.

I suggested they go home and read Marie Kondo's *The Life-Changing Magic of Tidying Up* and then KonMari the backlog. Remove anything that didn't "spark a sense of urgency" because if it didn't, it was destined never to get done. Not at this pace.

They agreed that there were many things that would never get done in the backlog, but they were not comfortable removing anything.

"What if we forget these things?" they asked.

"Wouldn't that be great?" I replied. "To be able to forget all the things you feel you *should* do, but know deep inside that you have absolutely no intention of ever doing? You are only keeping a record of your bad conscience! By keeping this backlog, you are in a perpetual state of being behind, creating the feeling that you will never, ever catch up."

While they understood my point, they still couldn't bring themselves to delete the backlog items.

We agreed to introduce "The Lake." In the Swedish version of the card game "Go Fish!" you say "Finns i sjön!" (it's in the lake) when the opponent is to draw a random card from the center pile. The Lake was a non-prioritized bunch of backlog items that, for one reason or another, had gotten stuck in limbo. These were not discussed during refinement but were not considered to be permanently deleted. Whenever we prioritized items in refinement, we had the option to call out "Finns i sjön!" and let the backlog item "swim freely in the lake" until we had capacity to take it on.

In the end, we never had the capacity to fish anything out of the lake.

When telling this story, it has been pointed out to me that our habit of calling out "Finns i sjön!" in this way is not strictly in accordance with the rules of Go Fish. But it was a good catchphrase, so it stuck.

Now that we've examined our incoming work, the importance of its structure for overview, and when to start, we need to look at when our work is finished: our Definition of Done.

Questions About Influx of Work Items

- Are we feeling pressured by our influx of work items?
- Are we in control of our influx of work items? If not, what causes us to lack that control?
- Are we able to focus on one task at a time, or do we need to switch a lot?
- Is our backlog filled with "guilty conscience"?
- Does the size of our backlog make us feel like we are always behind?
- Do we have time to finish our work in a timely manner with high quality?

Define "Done"

One of the main sources of unplanned work is the low quality of our output. A standard practice to make sure we do not release unstable or half-finished software to production is to establish quality measures in our Definition of Done.

A Definition of Done (DoD) should serve two purposes:

1. As a checklist to ensure all agreed-upon work is done before pushing to production and closing the work item
2. As a shield to ensure we allow ourselves the time to maintain the quality and stability of our software—and, in the long run, our work environment

When the pressure is on and time is of the essence, the latter purpose of the DoD tends to become negotiable.

Corners are cut. Test coverage and entropy control, such as refactoring and stabilization, are skipped. Documentation is either rushed or neglected. This is the software equivalent of getting drunk to forget your problems. It feels better for a short time, but the next day, you wake up hungover, and the problems are still there—maybe even in a slightly worse state than before.

Allow Time for Quality

When we skip quality assurance to "save time," we are really only borrowing time from our future selves. If testing is habitually skipped, it usually causes both pressure and complexity to increase over time, as more work is generated due to bugs or design errors. We can't push quality assurance aside for too long without risking losing control of the situation.

As mentioned, when something breaks and forces our hand, we are no longer in control of the workflow. Some bugs and breakages are unavoidable, but if they cause significant time and focus leakage, we should address them promptly.

A good DoD should protect us from short-sightedness. It should guide which tactical decisions to make to reach our strategic goals regarding quality and documentation. (This is, of course, presuming we *have* such strategic goals.)

The DoD should also be an agreed-upon baseline. It shouldn't be written to cover every possible edge case but to capture the lowest common denominator for most work. If your Definition of Done is perceived as overly cumbersome and difficult, you will most likely start to justify skipping parts of it or spend too much time and effort on unnecessary things.

Refine Your Definition of Done

I was working for a company where the DoD didn't apply to each and every Story (in many Agile organizations, an Epic is the overarching work package that contains several Stories), but it was part of the work done before finishing up an Epic.

This practice allowed developers to focus on coding and solving functional problems for a longer period of time. The consequence was a lot of stabilization, clean up, and test coverage to be done at the tail end before committing to the common codebase.

The DoD also served as a checklist in a joint review meeting between the development team, managers, and stakeholders, called an Epic Closure. Over time, things were added to the checklist, mainly as a response to a team missing vital checks on edge-case Epics. Sometimes, things were added because a single customer demanded it.

After a while, we ended up with a long and specific list of things to check and cover, even for the Epics where it wasn't strictly necessary or didn't bring much extra value. It was a case of continuously adding things, but the collective knowledge of why they were there had been lost. In the end, no one dared to question this cumbersome DoD because surely everything had been added for a good reason.

Since we were always under pressure to deliver new functionality and performance improvements to our customers, we were constantly looking for ways to shorten our lead times. One day, a colleague called me up. He was a stakeholder who sat in on many Epic Closure meetings and had noticed unnecessary efforts on DoD items.

"Our Definition of Done has become a recipe for boiling macaroni," my colleague said. "It's too specific, and it's impossible to easily understand what's important in it."

"A recipe for boiling macaroni?" I asked.

"Haven't you noticed?" he said. "Every packet of macaroni has a very precise recipe for how to cook it. 500 ml of water. 0.5 tsp salt. 75 g pasta. Boil for 7-8 minutes. Right? Very precise and specific. However, with experience, you learn that the only thing you need to focus on is the boiling time. You can use any amount of water, so long as the pasta is covered. You can add the salt *after* boiling should you forget it. And the amount of pasta really depends on how hungry you are. Right? But the boiling time is always 7-8 minutes."

"Yes, your point being?" I wasn't quite following.

"Right now," he continued. "We are being too specific in our Definition of Done. I see time and effort being wasted on measuring precisely 500 ml of water, for no good reason whatsoever. We can save loads of effort by revising this macaroni recipe!"

He was right. Our Definition of Done was causing an unnecessary workload. Items had been added to the checklist to cover some special cases that only applied to a small number of Epics, and we ended up doing them for everything. These special cases should've been covered in the acceptance criteria for that specific Epic instead of added to the DoD.

The point is, only include stuff in your Definition of Done that applies to all work—the rest goes in the acceptance criteria. And the first item on your Definition of Done should read:

> *All acceptance criteria have been fulfilled and verified.*

Figure out Your "Boiling Time"

When in a stress pickle, figure out the "boiling time" and make sure everyone understands that this is what we focus our efforts on to ensure control of the workflow. This might not require 100 percent test coverage. Simple sanity checks could be enough, at least to start with, to get basic testing in place. Perhaps employing system monitoring that catches performance degradations, to give a heads-up before things actually fail, is a quicker stress relief than making sure everything runs perfectly.

Once we gain basic control of our workflow, we can start addressing deeper quality issues.

To prevent quality from dropping, we need to be aware of when we cut corners and keep track of what we have "saved time" on. A common reason for cutting corners is the pressure of an approaching deadline.

We need to understand our behavior surrounding deadlines and maybe sometimes question how and why deadlines are set.

Questions About the Definition of Done

- Are we in agreement on the expected outcome of our work items/packages?
- Are we in agreement when the work is done?
- Are we getting a lot of work sent back after deployment to production, with requests for revisions and fixes?
- Are we able to use our Definition of Done as a shield to allow us to finish our work with high quality?
- Is our Definition of Done a macaroni recipe?

Manage Follow-up and Deadlines

Generally speaking, I am not a big fan of deadlines. Not the time constraint in itself, but how they tend to be either arbitrarily set—meaning nothing really happens if the deadline is overshot—or represent an unrealistic timeline based more on wishful thinking than the facts of reality.

Fans of the deadline refer to Parkinson's Law, stating that work expands to fill the time available for its completion. I agree that this can be true, but when deadlines are set as a means to increase performance, they act more as a threat than a focal point. For deadlines to be truly effective, we still need to master what I have mentioned about prioritization, structuring work, influx of work items, and the Definition of Done. Otherwise, we may just have created a self-imposed "time poverty"[7]—when we perceive we have more things to do than time to complete them.

Even though tight deadlines can increase performance in the short term, over time, they hurt quality and well-being.[8]

Only use deadlines when they actually mean something to create clarity, priority, and focus. Use them when there is a specific market window you need to hit or new legislation takes effect that you need to be on top of. This is what deadlines are for. Of course, sprint goals are a form of deadline, but these are set by the teams themselves to create clarity and priority: when in doubt, do whatever takes you closer to the sprint goal.

Try a Crunch Period

Some companies I have worked for experimented with crunch periods, when we'd put everything else aside and focus all efforts on getting one particular piece of the product out the door. These were often pretty successful in delivering new value, but we could only manage one or two crunches per year. So, they needed to be used wisely, with clear targets and wrap-up handling afterward.

When we forgot to discuss and decide on how to handle the wrap-up, the crunch tended to continue but with less momentum or fizzled out without any feeling of closure.

If you already have the ability to prioritize and focus on what is most important, skip deadlines where possible and trust that the work will be finished when it's done. Continuous follow-up helps to keep the priority and focus, which should be the main benefit of your daily stand-ups!

7. https://static1.squarespace.com/static/5e270ffe949f6c4d0d0fcc55/t/5f77c04124f75f5a9d823ba6/1601683526489/Why+time+poverty+matters+-+giurge+whillans+west.pdf
8. https://oulurepo.oulu.fi/bitstream/handle/10024/28429/nbnfi-fe2020041516665.pdf?sequence=1&isAllowed=y

To be able to have a decent follow-up, we need to be able to easily visualize our work and status—a well-maintained backlog with well-defined items. If stories carry on for weeks or months, without even a subtask or a checklist to follow status and progress, you are in the dark. And there is probably work going on that is not visible on your Kanban or Scrum board or Excel sheet, whatever you are using to track progress.

Never use deadlines as checkpoints or a means to finish more work in a shorter time frame. Only use them when they really, truly matter, and make sure they are used to clarify priority and focus. Clear the schedule of everything else.

As mentioned in the opening chapter, this is not an exhaustive list of examples of work structure-related sources of pressure; it is just some of the main ones that I have identified in my work throughout the years. You may have other sources in your context that you should address and explore.

But for now, we will leave the subcategory of Structuring Work to examine another source of Pressure: what motivates the individual.

Questions About Follow-up and Deadlines

- Can we easily measure or visualize our progress?
- In what way do our deadlines impact our sense of urgency in a positive way?
- In which cases do our deadlines impact us with a non-constructive increase of pressure?
- If we were to do a crunch period, what would we focus on delivering?

Motivate the Individual

Even though the whole idea behind writing this book was to shift the focus from the individual to the context and system in which we work, we cannot remove the individual altogether. The organization is, after all, made up of humans—quirky, peculiar, and perfectly human humans.

With the advent of AI, we need to stop referring to “the human factor” as something negative. Moving forward, we need to consider the human factor as an extraordinary asset—something which, in all frankness, should always have been the case. It’s the human factor that will set our organization apart.

Our humans will be the ones able to create and produce things that AI will not yet be capable of. We need to be able to release our inner motivation and peculiarity.

All the high-performing teams I have had the privilege to work with have had one thing in common: a positive attitude and a good team vibe. Of course, it's not all about having fun. There is more involved in creating a great team than simply cramming a bunch of happy extroverts into a room. That's what makes group dynamics so interesting! In my experience, teams with predominantly introverted people and a penchant for sarcasm can still have a collectively positive, outward-looking nature.

Don't Underestimate the Power of Humor

My observation is that the ability to have fun at work, and humor is an essential ingredient, allows for a wider range of emotions and opinions to be displayed without getting stuck in bickering or resentment.[9] When we can let our guard down, we are open to more input and ideas.

These high-performing teams also had a collective sense of responsibility, which in turn created a sense of urgency. If a team does not feel—or is deprived of—responsibility for its own success, they will not feel responsible for any failure either. This could, for example, be a dependency on another team or department with a completely different set of priorities. If work gets stuck in these dependencies and a deadline is missed, neither team is likely to feel responsible for the failure.

Creating an environment for empowered teams, as well as empowered individuals, is a key to unlocking performance.[10]

When reading literature on motivation, it's easy to think that inner motivation is the only true form. That inner motivation and *passion* is the only road to expertise, success, and happiness. While passion is a potent source of inner drive, we still need a bit of that boring structure. Otherwise, in the extreme, we become lust-driven or obsessive.

Without the concepts mentioned under Structure the Work, companies will have difficulty reaching their goals. Even though I maintain the importance of having the ability to say no to customers and stakeholders, it's usually their

9. https://scholar.google.cl/citations?view_op=view_citation&hl=en&user=kOGPNRUAAAAJ&citation_for_view=kOGPNRUAAAAJ:d1gkVwhDpl0C
10. https://www.researchgate.net/publication/6436621_A_Multilevel_Study_of_Leadership_Empowerment_and_Performance_in_Teams

demands that push product developers to get concrete and actually deliver something useful. We need that external motivation, otherwise, we may become professional navel-gazers, producing perfect stuff that no one else is interested in. In the end, it doesn't matter if our team has innovated the most energy-efficient, user-friendly, and aesthetically pleasing toaster if the company is invested in solving tomorrow's need for eco-friendly artificial fertilizer.

Be Aware of Who You Hire

Every manager wants to hire people with a strong sense of responsibility and passion for their work. But, to hire these people and then not provide what I described in Structure the Work is outright irresponsible. This is especially the case if we are hiring young people who might not yet have the professional self-esteem to place demands on their surroundings.

Responsible, passionate people are not always fully capable of making sound judgments of what a reasonable workload looks like. They want to deliver quality. They want to show they are capable and responsible by acing everything thrown at them.

Not providing a regulatory system for them is to construct a stress trap, and we risk burning out or scaring off the best people we have—those who we rely on to actually get things done. In the worst case, we end up with colleagues who simply don't care because caring becomes dangerous.

And to be honest, that's what we would deserve.

Align the Sense of Responsibility

Without an inherent sense of responsibility within the people who make up our organization, delivery and performance capabilities will plummet. If an employee shows a complete lack of responsibility, they should be isolated and confined to an office in the basement, with all meaningful work taken off their hands.

That said, if someone in a team of educated people shows a complete lack of responsibility, there is usually a misunderstanding of expectations. Most people, especially those who've gone through the trouble of getting a higher education, want to use their expertise to do something meaningful, to be part of something bigger, and to see their work have an impact.

There are, of course, those hopeless bad apples who simply don't function properly. However, they are very rare, and if you find yourself surrounded by them on a regular basis, I think you need to start addressing your own cynicism. There are professionals who can help with that.

In my experience, when coworkers stop taking responsibility for things, it's a sign of organizational or cultural misalignment. Either their autonomy has been taken from them, or other systemic constructs make it hard for them to take full responsibility for completing their work. There might be a widespread not-my-job culture, or there may simply be expectations that are too vague for anyone to live up to them—or even understand them.

Don't "Reward" Responsibility with More Work

If our organization has a culture of "rewarding" responsibility by increasing the workload and adding an ever-growing number of responsibilities for those who show responsibility, we have probably shot ourselves severely in the foot.

While the attention might initially feel uplifting for those who get shown extra "trust" through added workload, the quality of work will soon start to suffer when the same responsible people are constantly bogged down. That doesn't send the right signals. Those prone to ducking responsibility will be even more prone to ducking, and the responsible ones who do not duck will run the risk of burnout.

A too-great sense of responsibility can paradoxically be an Achilles heel for the individual and a low-quality quick fix for the organization that resorts to "throwing people" at systemic problems instead of solving them.

Look out for Signs of Performance Anxiety

If we as leaders want to hire people of high responsibility, we are, in turn, responsible for managing the pressure they are under—foremost by providing prioritization and de-prioritization of work.

Ambitious, service-minded team members with a high sense of responsibility—just the kind we want—often have a fair amount of performance anxiety, especially the younger ones. It's a common downside of that drive.

To hire such a person and then expect them to be capable of withstanding an entire organization's inability to structure, prioritize, and filter the workload is a massive act of wrongdoing!

You may end up bereaving them of their passion for their work.

Questions About Sense of Responsibility

- Is it clear to us what our responsibilities are, and how do we show that responsibility?
- Do we agree with our responsibilities, or do we think something should be added or subtracted?
- In what way does our sense of responsibility contribute to our sense of urgency?
- Are there cases where we show a lack of responsibility? Why?
- Are there cases where we take too much responsibility? Why?

Harness Passion

To work at something that is your passion is a privilege and a strong incentive. I am using the word passion in a positive sense here. In research on passion and its impact on work-life commitment and satisfaction, there is a difference between "harmonious passion" and "obsessive passion."[11] In short, harmonious passion is when you are in control of the activity and obsessive passion is when the activity is in control of you. We will keep it a wee bit unscientific and claim that passion for your job is a good thing.

Another unscientific claim would be that the more bureaucratic an organization is, the harder it becomes to rouse people's passion. Naturally, everything we do at work can't be what we find most enjoyable, but there has to be a critical mass of what we enjoy for us to keep our enthusiasm. Companies need driven, curious, exploring people to be able to excel, and there needs to be room to do that exploration without too much form-filling. Passion doesn't always seem to fit well into corporate templates.

There are a lot of different things companies can do to help passion along. For example, there might be hack days or innovation sprints, where employees can focus on whatever they think is the most important or most inspiring thing to work on. These are great practices, but I also think companies need to ponder why they're needed. Why don't our employees get this type of satisfaction in their everyday work?

Another great practice would be for companies to keep track of what the employees choose to do on hack days. What the employees choose to do when

11. https://psycnet.apa.org/record/2003-08110-016

given full autonomy of their time is perhaps a great indicator of what the company should focus more on.

Make Time for Innovation

I was working at a very innovative company at the forefront of machine learning. We introduced an innovation day once every quarter. We called it Innovation Friday, though it started at noon on Thursday and lasted for as long as people wanted on Friday afternoon. Everyone could choose whatever they wanted to do and whether they wanted to do it alone or team up with others. The only demand was that they wrote about what they had learned or discovered on our communal wiki, and they were encouraged to do a show-and-tell on our sprint demos if they thought it was of common interest.

Some chose to simply read a book or focus on developing their skills by doing programming exercises. Most decided to team up and experiment together. There were a couple of good breakthroughs discovered during these Innovation Fridays, but what was most striking to me was that almost every single time, one or two groups would come up with new tooling or scripts to make their own everyday life easier.

To me, this was a clear indicator that the company should have spent more time and effort on the engineer's working conditions. We should have looked deeper into what was eating away at their time and where we were leaking capacity.

When smart people have autonomy, they show you what they think is important. Not letting them work with what they think is important will cause them frustration. But, before we run wild with passion and autonomy, there is one more thing we need to mention: alignment. To make things work in a scaling environment, we need to have common standards.

Identify Where You Need Alignment

I have worked with teams having full autonomy on tooling, tech stacks, and ticketing systems and witnessed it becoming a mess one time too many to not grow weary of full team autonomy—especially if it's a multi-team setting.

We need to have a way to assess where a standard way of working is required because there probably need to be code and branching standards and a best practice for documentation. There should be enough alignment to allow for cross-team collaboration without there being a steep learning curve to consider. If possible, these issues should be solved through a Community of Practice.

If we intend to grow beyond one or two development teams, this is basic hygiene. The best way to keep clean is by allowing our experts to form, for example, a Community of Practice to make the decisions. It's a way to create alignment without a negative impact on autonomy and passion since it's the ones affected who decide.

So, we need passion, but we can't let it run away with us. Passion can blind us to parts of our work that we don't want to see—be that taking on an overwhelming workload or having idealistic convictions that don't translate well to a bigger context.

As with everything, there's a balance, but we need to make room for passion. If we kill our colleagues' passion, they won't grow and develop, and we will slow down and stagnate as a company.

Questions About Passion

- Which situation or behavior demotivates me the most at work?
- What constitutes work satisfaction for us?
- What contributes in a positive way to our work satisfaction on a daily basis?
- When do we wish we had more autonomy?
- Where would we benefit from having more alignment?
- If we had full autonomy, what would be the first thing we fixed?

Enable Personal Growth and Development

Things are moving fast these days, and tech is probably one of the fastest-moving businesses. Keeping pace can be quite daunting.

This is another reason why hack or innovation days are a good idea to introduce into your company's rhythm. It gives people the chance to focus on their own competence and development.

Getting time to grow and develop on company time is important. You need to invest your spare time to become great at what you do, but this shouldn't be something employers require. Employers need to take responsibility for the growth and development of their personnel. Even with the best intentions, when most of us get home from work, we're tired and have family commitments or hobbies that we want to—and should—spend time on.

Facing change and needing to learn and adapt can be very stressful for most of us.[12] If we are already under a lot of pressure from other places, we may not have the buffers needed to be open to change and personal development. When something has to give, that stupid new "way of working" will get the short end of the stick.

Personal development is a long-term thing. Having to put out production fires for years on end may cause us to fall behind and, in the end, resist change altogether because we don't have the capacity to deal with the pressure.

Apply Your Drive in New Areas

This topic was a major source of pressure for me as a programmer. I'm talking about the relentless, never-ending need to learn new techniques, programming languages, best practices, and frameworks that popped up—and disappeared—on the tech scene.

I was forced to face the fact that my interest in technology was not strong enough. What drew me to programming in the first place was the puzzle—the solving of the problem—and to make someone's day just a little better by improving or adding new features to the software. The technology used to do it was of secondary interest. This realization coincided with Team Cumulus's need to revamp and modernize our product, as we quickly needed to adapt to new technology and solutions.

It was a tough awakening for me because being a programmer is a very strong identity marker. It was who I was and what I had always wanted to be!

When we have a strong passion for what we are doing, personal growth and development within our field will come naturally. If that passion has waned, however, there might be a problem.

We are all responsible for our professional and personal development, but employers should facilitate and make room for us to take that responsibility. They can do this by setting aside time or providing input on what they see as important and allowing us the focus to learn new things.

It's a win-win no-brainer.

In the case of Cumulus, my employer gave me the opportunity to explore a newfound interest in Agile and Lean practices, as well as my interest in team building and organization.

12. https://www.zeuchsbuchtipps.de/wp-content/uploads/2014/07/2011-Dahl.pdf

Questions About Personal Growth and Development

- In what way is it made possible for us to learn every day at work?
- What would allow us to learn and develop more in our everyday work?
- Which—if any—planned changes in our organization do I feel pressured or anxious by?
- What specialty would our team benefit from deepening our knowledge in?
- What general topic would our team benefit from gaining more knowledge in?

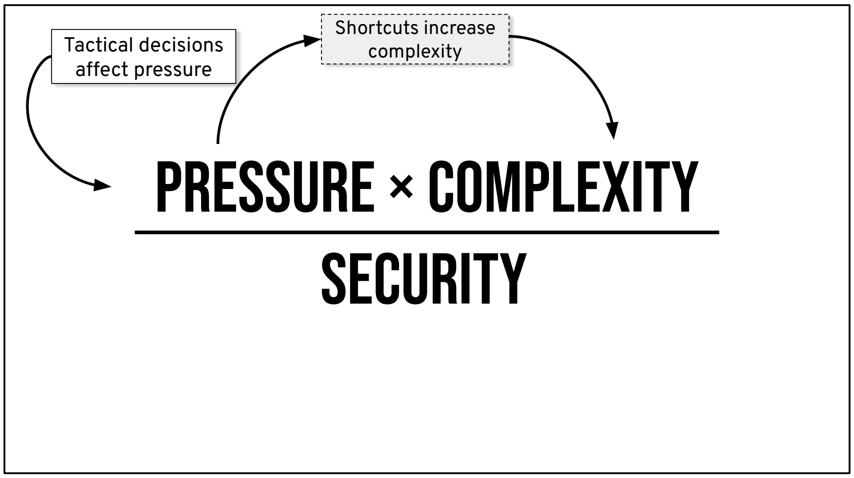
Wrap Up

Pressure is not a force of nature; it is decided by our tactical decisions. By tactical decisions, I mean how we decide what we should work on, how much work we have in our backlog, and how much time we have to get it done. In other words, how we prioritize, set deadlines, or decide when something is finished.

However, if we are struggling with quality issues and constant production hiccups, we are not in the driver's seat. When circumstances determine our actions, planning becomes a Sisyphean task that produces no value. At the same time, we need to be able to plan to get our workload under control. By addressing these issues, we can get back on track quickly, but we have to commit to doing so. A good start is to discuss some of the questions provided throughout the chapter.

And we *do need* to get back on track with this because organizations that are constantly under pressure will invariably develop an increasing problem with complexity. This happens because there is no time and energy left to look after our own house. When under pressure, we skip testing, skip refactoring, and take shortcuts. We will start taking *complexity credits* because that is really what "saving time" means.

So, if there is one takeaway I want to emphasize moving forward, it is that shortcuts increase complexity—as illustrated by the image on page 39.


Tactical decisions affect pressure
Shortcuts increase complexity
PRESSURE × COMPLEXITY
SECURITY

Before we take a deep dive into Complexity, let's visit team Cumulus again to learn how it began struggling with mounting production pressure and technical debt.

Remember

Pressure—the sense of urgency—is decided by:

- How we structure our work
- The motivation of the individual

How we structure our work includes how we:

- Regulate the influx of work items
- Decide our Definition of Done
- How we set our deadlines

Motivation of the individual refers to:

- Our sense of responsibility
- What we are passionate about
- Our personal need to develop and grow

CHAPTER 4

Team Cumulus: Building to a Break

If I have to criticize our development manager for one thing, it would be backlog handling. Things were not properly broken down and defined before work began. This meant tasks kept growing—either from scope creep or getting sidetracked. It was hard to visualize the status and progress of larger undertakings. We produced a lot of code, but we had little sense of where we were. Consequently, the team felt like we were swimming upstream in a river of maintenance tasks.

Part of the reason for this lack of structure was the ever-deteriorating relationship with our product manager. There was no development process for the huge reconstruction work we had to undertake, and he was not interested in putting one in place. He didn't want to be involved, didn't want to spend the production experts' time on it, and most of all, he didn't want to hear of any problems. If there were problems with the software, we must have caused them! Ergo, our problem, not his.

Team Responsibility

In all other aspects, Cumulus was a perfect team. We were helpful and service-minded, and we cared about each other. Within the team, we had psychological safety by the bucketful.

As pressure increased to add more customers to an already unstable system, the sense of urgency to replace Silverlight and the Fortran algorithms, stabilize the system, and modernize the software was palpable. We had been working under pressure for a long time, but at some point, without us noticing the exact moment, we passed a threshold, and our situation was no longer sustainable. We had always been able to stay in good spirits and maintain a helpful attitude, but circumstances were starting to wear on us.

When we started migrating functionality into our new tech stack, we essentially doubled the number of systems we needed to keep track of and maintain—while also learning the new tech. Production had to keep a business-as-usual pace, so all the old stuff needed to be kept up to date and running while we developed the new. This required a myriad of integration points, converters, and proxies for backward compatibility as we worked our way forward.

We had probably been a bit naïve in this regard. To tackle this beast, we needed more heads and hands to crank out code.

Off the top of my head, this is the tech rundown we would've needed to request people to have on their CV if we had been honest in recruitment:

Fortran	Bash	MariaDB
Java	SQL	MongoDB
C#	Silverlight	Postgres
C++	Jenkins	Visual Studio
Golang	Grunt	VIM
HTML/CSS	GitLab	Eclipse
TypeScript	Subversion	VS Code
JavaScript	Apache	Windows
Batch	RabbitMQ	Linux
Node.js		

You get the picture. We didn't mention all this in our hiring adverts, of course. Any sane candidate would have run for the hills!

Increase of Complexity

Since there was no telling when things might fail or start throwing exceptions, we had refined our CI/CD pipeline into a smooth-running machine. We could redeploy pretty much everything in a few minutes, often without the users even noticing.

As time passed, we spent considerable effort stabilizing the system to reduce the amount of surprises. What we couldn't fix, we placed under monitoring to get a heads-up on regressing performance in our services.

The team grew steadily, adding a new developer every few months as we ramped up to take on the challenge of rebuilding our product. Since we were doing a major overhaul of the user interface, we added a UXer to the team.

However, our non-existent development process was starting to become problematic. Our backlog had become this ever-growing bank of ideas in varying states of detail and refinement. Things were too complex to overview. It was impossible to keep track of the status, priority, and potential scope creep in what we were developing. We were in dire need of more structure and process.

We also desperately needed deeper collaboration from our product manager and production experts. Naturally, we didn't want to make a straight port of the old product. We wanted to focus on what was most important, make sure it was top notch, and preferably drop lesser-used functions while we were at it.

This was when things came to a head.

Our development manager had enough of the deteriorating relationship with the product manager and handed in his resignation. A short while later, the product manager left the company as well. Without going into details or pointing blame, the situation had become untenable, and something had to change.

In any case, we were ramping up development, so this was not a good time to be without leadership. We needed to get on top of the increasing complexity of our system. As long as we couldn't start shutting old functionality down, we were only adding to our workload.

Up until this point, we had taken on a lot of *complexity credits* to rush new functionality to production. We needed to step back and get the situation under control.

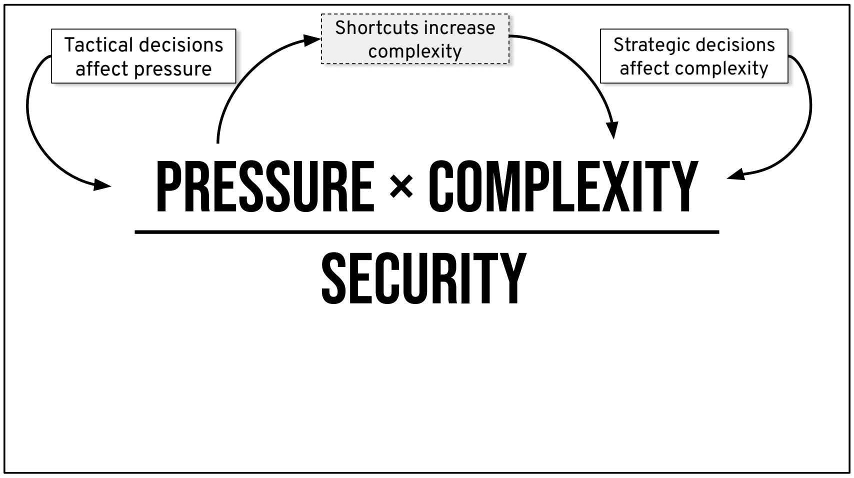
CHAPTER 5

Identifying Sources of Complexity

Mary Poppendieck, who coined the term Lean Software Development, has compared complexity in organizations to high cholesterol in the human body.[1]

You're probably not aware that you have it until you look for it. And if you're unlucky, it might kill you!

The following image illustrates how tactical and strategic decisions relate to the Pressure and Complexity variables of the Stress Equation:


Tactical decisions affect pressure
Shortcuts increase complexity
Strategic decisions affect complexity
PRESSURE × COMPLEXITY
SECURITY

Complexity tends to grow over time as a product[2] or an organization grows in size. If there are no strategies or company policies on how to counteract

1. https://www.youtube.com/watch?v=G32nUUMIXKE
2. https://users.ece.utexas.edu/~perry/education/SE-Intro/lehman.pdf

it, complexity can easily grow and become ever more costly and cumbersome. Strategies on product design, tech stacks, and infrastructure should be in place to provide guidance on when and how to act. They need to be clear enough to guide our everyday tactical decisions in the right direction.

Complexity is an important health indicator for all organizations and companies. The Stress Equation multiplies Pressure and Complexity because high complexity in combination with high pressure greatly increases the stress level. In contrast, lowered complexity will reduce stress even with unchanged pressure.

In this chapter, we will learn about different sources of complexity and how they occur. The aim is not to remove all complexity but to reduce the amount of unnecessarily wasted mental bandwidth. Complexity is affected by pressure, as studies show that under pressure, we are likely to make errors or take shortcuts, with lower-quality solutions as a result.[3] Similarly, stress has been shown to inhibit decision-making, where decisions are reached before reviewing all potential options.[4]

This is, once again, not in any way an exhaustive list of complexity sources—you may identify others in your context. However, in my experience, the main categories for these sources are either unnecessary cognitive load or frustration caused by system inertia.

To address complexity in a wider sense, we need to view it as anything that makes it harder for us to complete a task, finish work, and get stuff into production so it can generate user, customer, or monetary value. Technical debt might be one such slowing factor, as was the case for team Cumulus. But, it is not just technical complexity that slows us down.

Understand Your Complexity Debt

Technical debt is a concept that gets thrown around a lot, but I am not sure everyone who uses it actually understands what it means—or what the costs actually are. If that understanding was common knowledge, there wouldn't be so much of it. Also, the concept only catches the technical aspects of what slows us down.

This is why I've started to refer to it as *Complexity Credits*.

3. https://louisville.edu/psychology/decaro/lab/publications-2/beilock_decaro_2007
4. https://www.academia.edu/84354982/Decision_making_under_stress_Scanning_of_alternatives_under_physical_threat

Complexity credits are small debts that we accumulate over time. None of them are especially eye-catching on their own, but when zoomed out, we see a pattern of behavior where our complexity debt is increasing.

Acting on credit is a perfectly viable business decision. Sometimes we can't wait. If we don't have the financial means at this very moment, we call the bank to get credit. However, we can't continuously act on credit. If we don't have a viable repayment plan for those credits, the bank will eventually say no.

The problem with technical or organizational credit, as opposed to monetary credit, is there is no bank. We are free to take as many credits as we like. This will slowly build up cognitive load and system inertia—the interest rate of those credits—that stops us from being efficient. Eventually, this will reduce the speed of the organization and harm our ability to act under pressure.

Be Aware of the Type of Debt

While technical debt is probably the most obvious form of complexity credit in software, there are other things that cause us to accumulate complexity—both regarding our product or software and our organizational structure.

As our flow of work grows slower, there is a tendency to throw more people at the problem—coordinators, project managers, or more developers. This further deteriorates the system because more people always means more complexity. More brains to keep informed, more heads to keep aligned, more chefs that stir the soup—more meetings. To remedy the confusion that usually follows, we "increase clarity" by assigning ownership—essentially defining who gets yelled at when expectations are not met.

With this "clear ownership," people grow protective of their own interests and their own responsibilities. So, to get things done we need more negotiations—more meetings.

If this continues for years and decades, the organization will grow so complex that no one intuitively understands it. Official channels become so slow that it's impossible to get anything done. When this happens, there is a tendency for a "black market" of back channels to evolve.[5]

If this complexity is not addressed, knowing these back channels becomes the only realistic way of getting things done within a reasonable time span.

5. https://www.heinz.cmu.edu/faculty-research/profiles/krackhardt-davidm/_files/krackhardt-hbr-spring2011.pdf

Have a Mindset for Reducing Complexity

Alfred North Whitehead was a philosopher and mathematician, perhaps best remembered for writing *Principia Mathematica* together with Bertrand Russel. In his book, *An Introduction to Mathematics [Nor11]*, he argues that civilizational advancement is a matter of reducing complexity to preserve cognitive energy,[6] expressing it as:

> *Civilization advances by extending the number of important operations which we can perform without thinking about them. Operations of thought are like cavalry charges in a battle—they are strictly limited in number, they require fresh horses, and must only be made at decisive moments.*

In other words, we don't have endless cognitive energy. If we waste that energy on things that could be made simpler, we reduce our ability to excel at harder problems. How many companies have this type of policy when deciding on tooling, product design, tech stack, or organizational structure?

My take on this quote is:

> *An organization advances by increasing the amount of finished work without increasing the workload.*

With that in mind, what has your organization done in the past month to make it easier for you to deliver value to your customers? If the answer is *nothing*, you really should consider how you can start working with continuous improvement. Otherwise, you may soon find yourself in a very costly complexity spiral.

Listen to the Alarm Bells

One good indicator of organizational cholesterol is how long it takes for new employees to start delivering valuable work. How long does it take for them to understand the software architecture? How long does it take for them to understand the organizational context in which they will be working?

I have worked with companies that say it takes at least ten to twelve months for a new programmer to start delivering value. They almost say it with pride because they think they are so "advanced" that it takes an experienced engineer almost a year to understand the system and the product.

That shouldn't be a source of pride. That's an alarm bell!

Every company should be aware of when it is wasting its employees' cognitive energy. That energy should be preserved for the big puzzles and the real

6. https://quod.lib.umich.edu/u/umhistmath/AAW5995.0001.001

problems. Reducing cognitive load should be one of the main focus points for continuous improvement.

Questions About Complexity

- What should be easy for us to do, but is cumbersome?
- In what situation is it okay to let complexity increase?
- In which situations should we not let complexity increase?
- What, in the way in which we act, tends to increase complexity? How should we act instead?
- How long does it take for new hires to understand their surroundings and become self-sufficient? What could we do to make it easier for new hires and speed up training?

Reduce Cognitive Load

Very simplified, cognitive load can be described as the amount of information needed in your work memory to solve a specific problem or task—how efficiently you gain the knowledge needed to understand it. This mental workload can also be affected by context switching, in other words, how often you need to swap information in your work memory.

Problems that require a high cognitive load demand a longer time of uninterrupted focus for work memory to reach a state where it can visualize a solution—to reach a state of flow.

Most advanced software development requires deep focus for problem-solving. There are often numerous components and subsystems interacting, with varying degrees of straightforward integration. It's complex!

Reduce Context Switching

Interestingly, some studies suggest that interruptions do not necessarily have a negative impact on the speed or quality of work.[7] However, if the interruption requires a change of topic or context, it becomes disruptive, with negative consequences. In either case, interruptions come at the cost of increased stress, frustration, time pressure, and effort needed to complete the task. In short, the workload feels heavier.

7. https://ics.uci.edu/~gmark/chi08-mark.pdf

Some companies introduce meeting-free days to reduce the amount of context switching needed among their employees. These meeting-free days can still be used for get-togethers in order to collaborate on solving problems, but team members should be spared from recurring meeting series—specifically, meetings that are not directly related to the completion of currently ongoing work.

A team daily can still be held in the morning, because that is related to completing currently ongoing work. A staff meeting with information from management, however, is not related to currently ongoing work and should be avoided. I would also claim that any kind of status meeting should be avoided unless there is a significant need to synchronize efforts.

Avoid Status Meeting Hell

I coached a team that got caught in status meeting hell. A critical performance issue had been detected in the product, and the customer was going into a very costly integration test phase in just a few days. The issue needed to be solved yesterday!

Naturally, management and sales were very nervous—not delivering could potentially result in hefty fines, not to mention a significant loss of reputation. The team was working overtime and weekends to solve the problem, but they were constantly being contacted for status updates from various high-ranking managers. The senior developers were interrupted several times per day to give progress reports.

In the end, to save the team and get the job done, their line manager told them to gather in their team room and shut Teams and Outlook down. She then placed herself by the door and stopped anyone from entering. All communication had to go through her.

The problem was eventually found and solved in time for the customer to do their integration tests. I am convinced that had it not been for that brave line manager, this would not have been as successfully accomplished.

Manage the Management Meetings

Context switching is a drudge for a lot of employees. In this example, it was developers but this context switching is especially common in management. Managers are called into meetings left and right and get late calls for budget updates while they're in the middle of something else.

In talking to an Agile Coach at a company where I had just given a talk on the Stress Equation, she realized that her stress was caused by being part of too many contexts. She had been feeling performance anxiety because her work calendar was not filled to the brim, but she was still feeling stressed out for some reason.

“It’s not that I am pressed for time, but my cognitive load is maxed out. I simply cannot take on more contexts!” she very wisely concluded.

Now, before you paint me as a stout opponent of meetings, I have to say that I am not. I really like a good meeting! But, many organizations suffer from inefficient meetings with questionable purpose, vague outcomes, and bad facilitation. These meetings are boring at best, but at their worst, they increase the cognitive load without creating any value.

This context switching due to meetings is hopefully not what’s weighing heavy on our developers. Most developers are not meeting enthusiasts; in fact, it can be tricky to motivate them even for the important ones. If context switching due to meetings is a problem for your developers, you really need to get your priorities straight!

Code like Agatha Christie

What normally becomes a cognitive load for developers is problems with code quality, architecture, and documentation. Since this is not a tech-oriented book, I am not going to go into those subjects in detail. But, I will say this:

Code is like poetry. When beautifully written, with an impressive show of language skill, it can bring about powerful effects using few and fancy words. The problem with poetry is that it rarely has mass appeal. Most people don’t understand it, and they don’t enjoy it. It takes too much thought to process, so they can’t be bothered.

Good production code doesn’t need to be impressive; it needs to have mass appeal. It should be less poetry and more Agatha Christie—enjoyable to read and very predictable.

We can write poetry in our spare time. The code we write at work is for others, not ourselves.

That’s as much tech talk as you’ll get out of me on this topic. Instead, we will take a look at other, non-tech factors that can and will cause an increased cognitive load in software development—and maybe in other businesses as well.

Questions About Cognitive Load

- What drives cognitive load in our workdays?
- Is the number of meetings at a reasonable level?
- Is context switching a tangible source of cognitive load for us?
- What interrupts us in our workday and impedes us from focusing on finishing ongoing work?

Simplify Product Design

Good software teams discuss code and architecture frequently, in some cases, on a daily basis. If the code is hard to understand or architecturally difficult to overview, work soon becomes slow and inefficient due to high cognitive load.

However, it is not just the code that is at play here. Product design has a major impact on the technical design. If our offer is hard to understand, the software architecture is likely messy as well.

Many teams don't quite grasp the product they are working on because ever-more farfetched tweaks and additions are getting shoehorned in for individual customers. The code and the architecture usually fall apart over time when there is no red thread in the offer and product strategy. What are we selling? Do we have a clear vision of what the product should be?

Be Cost Aware

The costs of single customer customization is something a product or portfolio manager should be aware of.[8] It is not just the direct investment in development time but the added complexity when tailored updates are made for individual clients.

If we tend to sell these tailored solutions on the cheap to customers, there is usually a habit of adding new features and integrations "quick and dirty," taking code and architectural complexity credits as a result. Even with the best intentions, when things need to be solved fast, new additions tend to make little structural sense if you take a step back and look at them.

Customer focus is very important, but if we actually buy into the saying, *"the customer is always right,"* we will eventually find ourselves in a pickle! Customers probably expect a stable and affordable product. For that to remain the case, the product must be simple to maintain. Otherwise, it will pretty soon be both unstable and expensive.

8. https://cutlefish.substack.com/p/tbm-2152-building-stuff-to-close

This added complexity from product design neglect is one reason it takes so long to onboard new hires. It's not because the product is advanced—it's because it's a mess!

Have Design Conversations

There needs to be a product strategy guiding our decisions so our complexity doesn't increase more than the potential revenue. This requires a constant dialogue between sales, product management, and software developers.

If we do not have this continuous dialogue, we will rely solely on individual developers to raise concerns. But unless we have a habit of open dialogue on design topics, my experience tells me these concerns will likely be shot down by a well-aimed sales pitch.

We need an open and ongoing dialogue to be able to understand each other's stakes and needs regarding design decisions.

The same open dialogue should apply for our choices and efforts regarding the tools and support systems we acquire or develop. These should focus on making our lives easier, not be a source of extra work or puzzling interfaces.

Questions About Product Design

- What is the product? What are we selling? Why do customers choose us?
- What should be our highest priority every day to avoid losing sight of the fundamental idea of our product or service?
- What do we find hard to understand about our product or services?
- In which situations do we tend to take shortcuts that, in the long run, will increase the complexity of our product?
- Where do we have unnecessary complexity in our product or software that we really should fix?

Invest in Infrastructure and Tools

Have you ever been forced to work with an internal tool that is slow, hard to understand, or requires ten clicks for even the simplest of data entries?

The tools we choose should facilitate and ease our workday. They should make things smoother and not be a source of waiting time and frustration.

Here lies one of the great paradoxes in software development. In most cases, we are very thorough and particular about what we deliver to customers—the customer experience. What we offer our own personnel does not always get the same attention. When customers voice requirements and requests, we see potential revenue. When our own people do the same thing, we see potential costs. At least, that's my speculation after trying to make a case for infrastructure and tooling investments in the past.

When employees complain about slow servers or an awkward support system, responsible managers want to know how much it would cost to upgrade the server or change the tool. If we ask that question, we must also ask, what will it cost to *not* upgrade the server? Otherwise, we can't make an informed decision because we only have half of the scenario estimated.

Estimating the cost of not doing something is tricky because it's usually hard to quantify. What does an increased cognitive load cost? What is the cost of employee frustration?

Provide Adequate Infrastructure

There is a Swedish word that doesn't quite translate into English: dumsnålt. Dum means dumb, as you might have guessed, and snålt means tight-fisted or stingy. Dumsnålt means to choose something cheap but which barely fits the bill. It's something that won't perform the task efficiently and might end up useless or costing more in the long run.

Too many companies go for dumsnålt when shopping for infrastructure or support tools.

I worked in an engineering company with a very tech-heavy product. It required heavy-duty servers to build and test. Server queues of up to three hours were not uncommon. On a bad day, hundreds of engineers were waiting for their code to build so it could be tested. Nevertheless, we were constantly denied money to buy more server power because that was too expensive. At the same time, we were tasked with hiring at least fifty new engineers in the upcoming six months.

Add more engineers to an already faltering infrastructure? Really? Maybe hire one less engineer and upgrade the server park with the budget savings? No?

Dumsnålt.

It got so bad that people didn't mind working weekends because then they had the build servers to themselves.

Budget for Performance

The problem here would appear to be about budgeting. We don't see the saved time in dollars or euros because the personnel cost is still the same, regardless of how fast our servers are. The cost of the added server power is, however, clearly visible in the budget sheets.

The frustration and loss of workflow that waiting times create are also hard to measure in dollars and euros. This is why our policy needs to be that our most valuable asset is our people's time and cognitive energy. These need to be treated as a precious resource. If we have the opportunity to save our personnel time and cognitive energy, that opportunity should be seriously considered—and preferably seized!

In reality, there are limits to how far you can get in saving time and cognitive energy. But, without the guiding principle of conserving time and effort, our employees' ability to excel will suffer.

It's about allowing employees to get time for deep focus by reducing the time spent on interruptions. Speaking of interruptions, it's time to revisit a favorite subject: Influx of Work Items!

Questions About Infrastructure and Tools

- Which tools do we use today that work well? What is it about them that makes them good?
- Which tools do we use today that don't give us the support we need? What is lacking in these tools for them to support us properly?
- Is our infrastructure fit for purpose, and is it capable of handling the workload? If not, when does it become slow? Where do waiting times occur in our infrastructure?

Control the Influx of Work Items

Hang on, Marcus, you've already talked about the influx of work items in the chapter about pressure!

Yes, but the influx of work items can affect both pressure and complexity. If the influx of work items is unpredictable or you have a lot of different places where work can originate, complexity increases. It leaves you with less control and an increased cognitive load, which tends to leave a mess in its wake.

In software, two main sources increase complexity through the unpredictability of places where work can originate.

The first one is errors and bugs—what you in Lean would call failure demand.

Reduce Failure Demand

Failure demand is a good indicator of complexity due to quality issues. It's the amount of work created by our products not working as expected. If a service suddenly crashes, there might be fines associated with not fixing it quickly. This increases both complexity through unpredictability and pressure by the sudden introduction of a new, very short deadline. Not to mention increased cognitive load by context switching, and the increased pressure of having to drop other work, which in turn might have a deadline of its own.

It's not unusual for DevOps teams to have their Dev all but completely seized up due to overwhelming Ops workloads caused by instability, low quality in code, and neglected administrative functionality. The pressure is on for the team to deliver new functionality while they are stuck in a loop of putting out fires.

In the worst case, management will think it's an understaffing problem, so they hire more developers to increase output. This leaves the DevOps team even slower because now they have to onboard new colleagues and put out the fires. In the long run, this increases the risk of one or more senior developers getting fed up and quitting.

It's a far too common scenario, and while management is not necessarily wrong to think it's a matter of understaffing, we may be falling into the pattern of throwing people at the problem. The long-term issue might be understaffing, but the short to middle-term issue is more likely to be overcommitment. Overcommitment will force us to take shortcuts and build up technical debt—doing business on complexity credits.

If we are already in technical debt when launching brand-new functionality, I shudder to think of what the legacy systems must look like!

Focus on Quality and Automation

I was brought in to coach a DevOps department that had taken many shortcuts in order to reach a very hard deadline. The decision to take shortcuts was totally understandable because not launching on a particular day would potentially have cost them their entire business, with years of investments going down the drain—not to mention severe disruptions for their customers.

However, after launch, they decided to focus on delivering more new functionality that had already been sold to customers instead of cleaning up the shortcuts they had taken. This caused ever-increasing complexity in an already very complex system, where firefighting had become the main activity.

Some of the shortcuts decided on were to skip automation and providing administration functionality for some routine tasks. This lack of admin interfaces meant that first-line support could not perform vital tasks, leaving second and third-line busy with work that first-line should have been equipped to handle. This severely impacted their capacity for new development and refactoring to fix the problem.

To sum up, they had taken a lot of complexity credits to "finish" on time, and now they were paying the interest rate on those credits.

Product organizations don't always fully understand the costs of the shortcuts they take. If we don't continuously invest time in maintenance and complexity governance, we will eventually lose control of the situation. The consequences will be increasing lead times on new features, frustrated employees, and more friction and conflict when things begin to run late, with potential people and culture problems if left unaddressed.

This was the situation for team Cumulus and one of the reasons we needed a major shift in technology. We had reached the end of the road with the old tech stack we had invested in and were largely stuck firefighting in the legacy systems. The workload of simply "keeping the lights on" was so big that new development slowed to a crawl. This, in turn, increased friction and conflict in the organization since we couldn't deliver on what had been promised to customers.

Be Aware of Your Back Channels

The second source of unpredictable influx of work items is caused by those men (let's face it, it's mostly men) who sidestep the backlog process by approaching developers directly to "just get this thing done." These guys often pride themselves in being The Fixer. The guy who knows a guy—he who "makes things happen down in IT."

In small amounts, this doesn't pose a problem. In fact, it may be a good way of creating some goodwill and showing service-mindedness. However, this behavior becomes a big issue for organizations that already have problems with finishing things.

When it becomes an established fact that the only way to get things done efficiently is to circumvent the regular process—use back channels—you have a very troubling behavior to curb. In the worst cases, it puts the backlog totally out of play, with a significant amount of work being done under the radar. By then, management control is truly lost, as is overview and the ability to do meaningful planning, follow-up, or prioritization. This becomes an invisible complexity created by back channels.

Structure and control of the influx of work items are among the most important things a development organization—especially DevOps, with such a mixed bag of work—needs to master. So many things are riding on it.

Have a clear and communicated process or method for how work reaches your teams. Ensure it is followed. Don't accept the excuse, "but we are working Agile!" (Yes, I have heard that argument more than once.)

Agile does not mean willy-nilly. Agile takes discipline.

When an organization starts to rely on the fixers, it could be a sign of system inertia. When our official organization and established processes grow too rigid for work to flow efficiently, there is an increasing need for a black market of work being done through favors and connections.

This becomes a whole source of complexity and complications because within that black market, a whole new currency of competence starts building up. People will gain influence by their ability to navigate the informal system. They are unlikely to want to give up this power. So when we try to address the inertia by improving our processes or changing the organization, they are less likely to be onboard since the change is causing inflation in their competence capital.

In other words, the reaction to the initial system inertia becomes a new source of system inertia.

Questions About Influx of Work Items

- What are the sources of our work items?
- Where and how does unplanned work arise?
- Are we aware of the cost/effort of our failure demand?
- Which work items tend to "come in from the side"? How should these be handled?
- Do we allow a guy who knows a guy to "make things happen down in IT"?
- When is it okay to let a guy who knows a guy "make things happen down in IT"? And when is it not okay?
- Are we understaffed or are we overcommitted?

Address System Inertia

Peter Senge's book, *The Fifth Discipline [Sen94]*, emphasizes the importance of building learning organizations that can continuously adapt and improve. He describes the resistance to change as the result of entrenched beliefs, norms, structures, and processes that maintain the status quo.

System inertia is often used to describe an organization's reluctance or resistance to change and adapt. If we cannot adapt, our system inertia will grow into a general resistance that slows us down as we lose the ability to quickly remove blockers that impede our value flow. These impediments can be conditions or practices that steal our time and attention away from creating value and doing meaningful work.

Let the Work Flow

To refer once more to Mary Poppendieck's quote about complexity being organizational cholesterol, I like to describe finished, value-adding work as the oxygen of an organization. It's what makes a company vibrant and energized. With added cholesterol to the system, "worxygen" flows slower and less efficiently, making the organization tired, slow, and sluggish.

I say "value-adding work" because we are not talking about work generated by internal bureaucracy, which is the stuff we do to please the process.

An inability to remove blockers and bureaucracy will occasionally put a company with high system inertia in trouble when a particularly urgent piece of work needs to flow through the system. This is when we bring out the adrenaline shot, manifested as a task force. In a crisis, most organizations can pull together to work toward a common goal and deliver quickly by clearing bureaucracy and gathering all necessary competence to finish the work efficiently.

This practice puzzles me a bit because it shows we can if we have to. What is it that makes it so hard to pull together in an everyday, run-of-the-mill situation? Why can't we design our organizations with clear goals and less bureaucracy when it clearly impedes us?

Explore Subtractive Transformation

Part of the problem is probably our human preference for adding things when solving problems. The human brain tends to default to additive transformations rather than subtractive transformation.[9] When designing processes and

9. https://www.nature.com/articles/s41586-021-03380-y

organizations, it's easy to understand how we would be tempted to add another step or layer of control to clearly show that we have done something about the problem—to prove that we have thought long and hard about this, to catch every edge case and theoretical eventuality. Even worse, if someone who is no longer with the company introduced a process step, we might not know why that step was introduced in the first place and might consequently not feel comfortable removing it.

Over time, we see a build-up of complexity credits, consisting of largely unnecessary processes and bureaucracy. And, as anyone who has ever been in serious debt will tell you, once the debt has built up, it becomes an ordeal to get rid of!

Questions About System Interia

- What are our biggest time thieves that we are *not* in control of?
- What are our biggest time thieves that we *are* in control of?
- What do I consider to be the biggest blocker preventing me from taking control of my workday?
- In which situations do we tend to add solutions when we should really remove problems?

Create Enabling Processes

Elon Musk has a pretty aggressive stance on process constraints when building rockets at SpaceX.[10] He has a solid understanding of the costs of accumulated complexity credits. For example, whoever introduces a constraint in the process owns it personally. This makes it easier to challenge the process step and prevent the loss of knowledge about why it was introduced in the first place. And process steps are being challenged all the time at SpaceX. In fact, Musk's general rule is that if you don't end up adding removed steps back 10 percent of the time, you are not deleting enough from the process.

In Agile, you are almost forbidden to talk about processes. You're supposed to talk about methods and ways of working. I think there should be more room to talk about the process. Processes aren't evil; in fact, they're quite useful when employed the right way. Maybe this is just my Lean-brain wanting to standardize the world in order to measure, follow up, and find bottlenecks, but I am quite fond of a good process. We all have our biases, I suppose.

10. https://www.youtube.com/watch?v=t705r8ICkRw&t=805s

The ability to evaluate is important for companies. If we can't evaluate, we have no basis for action. If we have no basis for action, we may find ourselves unable or unwilling to make decisions since there is no way of making informed prioritizations. A process should provide the basic structure for measurement and control.

However, while we need structure and control, measurements for evaluation should never become the actual purpose of the process. We need to continuously ask ourselves if our process is devised to facilitate efficient and effective work or just measurement and control. Our most precious resource is our employees' time and cognitive energy, so the process and measurements must be as simple and non-disruptive as possible. Preferably, they should be so simple that we don't even think of them as process steps at all but more like…a way of working!

Oh, dagnabbit! The agilists were right!

Figure out Where Alignment is Needed

Jokes aside, even when striving for autonomy, we need alignment to keep things from derailing, and process can be a way of aligning our efforts. This is one of the major challenges when scaling a company in size. When things are small-scale, we can be a bit loosey-goosey and informal. Once things take off and start scaling at a high pace, loosey-goosey becomes a problem.

We need to figure out where we need technical alignment and a common way of working. This is where we potentially encounter resistance because it means someone will need to give up autonomy in favor of alignment. Those who enjoy working in small-scale, loosey-goosey contexts may not be as excited by the prospect of things turning more corporate.

I am not trying to pass judgment on anyone. We are all different and like working in different environments and contexts, and I am personally not a fan of things scaling into corporate. But we need a way to evaluate and regulate. Call it a method, model, way of working, process, or whatever floats your semantic boat. Whatever it is, it has to be non-intrusive, simple, and preferably enabling rather than constraining.

That may sound like a tall order, and that's because it is! This is really hard stuff to get right. But if we don't find a way to do this in a lightweight and easy-to-understand way, we will see a growing need for coordinating and regulating roles that have the sole purpose of nagging about the process and filling out meta-documents—documents not directly related to finishing the work, but to please the process. This is the sign of a burgeoning bureaucracy.

When writing that, I suddenly realized that my job as an Agile Coach sometimes entails nagging about the way of working. To quote a team I once coached, when trying to explain why they hadn't done their backlog refinement the way we had practiced:

"You didn't nag enough, Marcus, and maybe we didn't listen enough…"

In the Agile Coach's defense, that nagging should be related to enabling teams to take responsibility for their own delivery by reducing the need for coordinating and regulating roles. The more things we are able to take joint responsibility for, the more overall responsibility everyone will feel—hopefully minimizing the need for future nagging or explicit individual ownership.

Questions About Process

- Do we have an established way of working?
- Do we need an aligned way of working? If not, why? If so, in which cases?
- In which cases do our processes give us good support in our daily work?
- In which cases do we not follow our processes? Why don't we?
- When is it okay to circumvent our established way of working?
- When do we tend to disregard our way of working when we really shouldn't?
- What do we wish we measured and followed up on that we don't do today?
- What do we measure and follow up on today that we really shouldn't?

Avoid Roles with Sole Ownership

Role clarity is an important aspect of setting expectations[11] for employees. This has proven to be important for work satisfaction and performance.[12] But, we need to make a distinction between expectations and ownership, especially

11. https://www.researchgate.net/profile/Shahidul-Hassan/publication/263301265_The_Importance_of_Role_Clarification_in_Workgroups_Effects_on_Perceived_Role_Clarity_Work_Satisfaction_and_Turnover_Rates/links/5efb3ccaa6fdcc4ca43db92d/The-Importance-of-Role-Clarification-in-Workgroups-Effects-on-Perceived-Role-Clarity-Work-Satisfaction-and-Turnover-Rates.pdf
12. https://journals.sagepub.com/doi/10.1177/014920630002600104

when sole ownership risks creating a not-my-job type of attitude. This issue usually becomes apparent when problem-solving is slow because we spend more energy figuring out who owns the problem than solving it. No one wants to volunteer for fear of suddenly owning another problem.

People sometimes protest when I claim ownership can be a folly because they want to know what is expected of them. This is absolutely fair, but expectations can be communicated without explicit ownership. For example, when I act as a Scrum Master, I expect everyone in the team to understand how our Scrum Board works and feel ownership. As a Scrum Master, I don't own the Scrum Board, but I am expected to ensure that everyone in the team understands how it works so they can take joint responsibility for keeping it up to date. Otherwise, I can never go on holiday!

Clear expectations, yes. Clear ownership, not necessarily.

Roles with sole ownership should be created and assigned cautiously because once a particular role owns "something," it removes responsibility from everyone else in the organization. Over time, the organization loses the ability to handle that "something" on its own, and the role becomes essential for the organization to operate. When that happens, removing the role takes significant effort because you have to retrain the organization to take back the ownership and responsibility of said "something." This becomes a source of system inertia.

This is why relying on sole ownerships can be folly. We should aim for collaborative responsibility as far as possible, and we most certainly shouldn't create new roles and positions to hide larger problems.

Don't Switch Buckets, Fix the Roof

In my experience, new roles are created when there is a mess to handle—something that needs cleaning up. Instead of taking a step back to do a root cause analysis, a role is assigned to handle the symptom—to clean up and regulate the mess. This is like placing a bucket under a leaky roof instead of fixing the hole.

The role gets tasked with periodically switching buckets as they get filled and risk overflow. The role then needs to join the weekly Messy Floor Committee to report on bucket status. He presents the Bucket Process that has been put in place and requests the purchase of bigger buckets to extend the period between bucket switches. The role gets complimented on a job well done. Not a single overflow in the past six months! This is clearly a competent individual, perhaps eligible for a promotion to Head Bucketeer?

After a few years, a consultant steps in and is tasked with reviewing and optimizing workflows. He immediately spots the bucket and asks why the roof isn't fixed to remove this manual task that ties up competent people. The bucket is clearly a more expensive solution in the long run. The Head Bucketeer feels a sting of indignation. That leak has always been there, and the Bucket Process has worked flawlessly for ages. This is not a problem. The Messy Floor Committee agrees; there has not been a mess at all since the Bucket Process was implemented. The consultant should focus on the real problems instead.

This story may sound Monty Pythonesque, maybe even a little bitter, but it's not far from how discussions go in organizations with a habit of focusing on symptoms rather than root causes. This behavior is harmful to organizations because they don't practice becoming better. Instead, they hide symptoms, which impedes the ability to improve the overall system.

An organization that defaults to assigning roles to handle symptoms risks losing its ability to fix problems at the root cause. It also creates resistance from people with a perceived important role since they are likely to defend their position, not wanting to give up what has brought them influence and prestige.

Be wary of roles that are assigned to hide symptoms. Don't settle for the bucket. Fix the roof!

There are situations when things need to be handled quickly because the mess is an impediment or a risk. In those cases, the role should be tasked with fixing the immediate mess and then given authority to fix the root cause. There should be a clear expectation that this role is temporary and part of the job is to phase itself out.

Regarding ownership, in Agile, we often emphasize the importance of autonomy, but there is a balance to be struck between one's own autonomy and its potential impact on others. When autonomy becomes self-sufficiency, collaboration over organizational boundaries will likely decrease, and you risk creating an internal monopoly.

Avoid Internal Monopolies

In the book, *Six Simple Rules [MT14]*, an internal monopoly is described as any organizational unit that affects what other organizational units can and cannot do. The internal monopoly has the power to create its own rules and tends to stress the importance of these rules to legitimize itself to the point where they become bureaucratic.

Without the intention of getting political, I find it an interesting paradox that, as a capitalist society, we normally think monopolies are bad while free and fair competition is good. By avoiding monopolies, we achieve a balance for consumers between the cost and utility of goods and services. If a consumer is unsatisfied with the cost or quality, they are free to choose a competitor. That market freedom is something most large companies lobby to protect or increase.

Paradoxically, the opposite appears to be true within the walls of many large companies. Few organizations have an internal "free market." On the contrary, large organizations tend to create internal monopolies—often under the guise of economies of scale.

For example, no company centralizes IT to increase user satisfaction—in this case, employee satisfaction. The priorities are usually:

- Cost reduction
- IT security
- Standardization

It's hard to object to any of these at face value when proposed, but depending on how they are implemented, the outcome might become the opposite. Cost reduction at any cost. IT security by protecting the company from its own employees. Standardization to the point where no one gets what they really need. (Dumsnålt!)

Centralized IT tends to lead to one-sided technical interpretations of policy to reduce the workload and costs of the centralized unit. The butterfly effect of this is that someone else needs to take the workload and adapt to this one-sided interpretation.

In the long run, this leaves ample room for resentment and indifference, which is a fertile growing ground for frustration and a not-my-job attitude.

There are significant benefits to centralizing IT, so I am not knocking the practice altogether. But, centralization usually means moving the doers away from the users. You need to be aware of the potential trade-offs and very clear on why the centralization is made and which potential negatives should be avoided.

Given self-sufficiency and the wrong KPIs, an internal monopoly can create severe constraints that will not serve the well-being of the entire system. It is free to suboptimize for its own performance goals and may start to exhibit silo behavior, which will harm the flow of the organization.

Questions About Roles and Ownership

- Is it easy to understand what is expected of us?
- Which areas of responsibility feel fuzzy to us as they are today?
- Which roles ought to be full-time positions?
- What stops us from taking joint responsibility for X? (Where X is an area of responsibility handled by a role today.)
- Which roles should we work on phasing out?
- Where are we settling for switching buckets, when we should be fixing holes in the roof?
- Where do we have obvious conflicts of interest in our organization?
- Toward whom—department, unit, or function—in our organization do we feel animosity or resentment? Why?

Focus on Flow Optimization

Organizations that focus on suboptimizing individual departments instead of their systemic ability to deliver value—finishing started work—will see an increased need for coordination and juggling as the number of ongoing projects increases. It may sound obvious that you shouldn't start more work than you can finish, but in reality, this happens all the time.

Without the ability to finish ongoing work, the number of parallel initiatives and activities will become increasingly harder to overview. There will be confusion and ambiguity in priority and expectations. We might end up spending more time and energy coordinating. Stress is often abundant in these coordinating roles when they need to rely on departments and people with conflicting priorities.

This is most common in organizations that have low value stream consideration in their design, so that work needs to cross several department lines in order to get finished. Just as an example, I once had to wait four months for the centralized IT department to set up a web server so that my team's stakeholder could test a product demo on the intended users. Our demo was nowhere near their top priority, and we were not allowed to set it up ourselves.

Identify Macro Level Constraints

On a macro level, as Eliyahu Goldratt describes in *Theory of Constraints* *[Gol90]*, we need to be clear about what purpose each organizational unit serves for the overall functionality of the system. Then, we can identify the constraints and where we have impediments to reaching the organization's goal. For example, we can identify where queues build up and cause waiting time for other organizational units.

We need to understand this because the slowness of the organization as a whole will affect the micro level—the performance of our teams.

When external dependencies are slow to react, there is no gratification in trying to move blocked work forward, only resistance. When faced with this situation, software teams tend to gravitate toward smaller work that is within their control—things they can finish themselves without external involvement. I have seen this in teams in several places and have displayed the same behavior when working as a programmer. To get at least *some* kind of work satisfaction from actually finishing *something*, we easily get stuck bug fixing or pottering about in the codebase, telling ourselves we are refactoring.

This is because we feel more motivated by a sense of progress. Completing bug fixes or random acts of refactoring provides a sense of accomplishment and boosts self-efficacy—our own belief in our ability to succeed.[13] Complex tasks with low momentum, especially with external dependencies, might not offer the same sense of progress, leading to decreased motivation and negatively impacting the flow even further.

Structure Work for a Sense of Progress

When coaching teams, I stress the importance of well-broken-down and structured work packages. It's good for a whole range of reasons, one of them being that we are more likely to identify external dependencies early and can sync with them so they are ready to help us when needed. Even with competent, cross-functional teams, situations will occur when we have external dependencies to handle.

Another reason is for the developers' well-being. You might not get to finish the whole user story in one day, or even a week, but maybe you will get the satisfaction of closing a couple of sub-tasks or ticking a few boxes in a checklist every day, which creates a sense of progress and momentum.

The brain prefers a small reward now rather than a big one later. This phenomenon is called temporal discounting. The human brain, in all its fantastic splendor, is still dumb enough to feel pleasure from crossing things off a checklist.

Make good use of your dumbness!

13. https://econtent.hogrefe.com/doi/abs/10.1024/1010-0652.21.3.241

Questions on Flow

- Is there any particular type of work that tends to get stuck or blocked?
- When our work gets stuck, what can we do to unstick it?
- How can we avoid our work getting stuck in the first place?
- Which work do we have that should have been done a long time ago but wasn't? Why wasn't it?
- In which situations do we tend to start new work when we really shouldn't?
- How often are we "refactoring" rather than refactoring?

Wrap Up

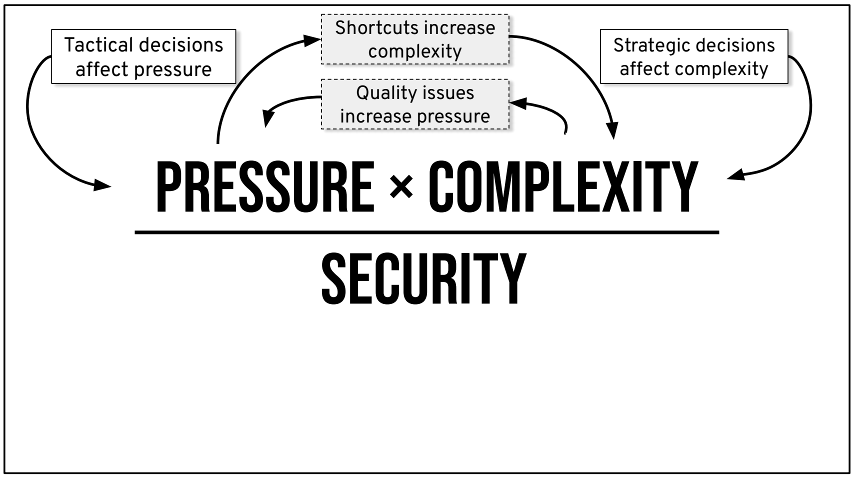
Complexity is, if put very simply, whatever impedes us from finishing the work at hand. Strategic decisions affect complexity, or perhaps more accurately, a lack of decisions and poor follow-through tends to increase complexity. Things like product design, quality levels, and organizational structure are all down to strategic decisions.

In contrast to tactical decisions, strategic decisions take longer to take effect. They might even require some degree of stamina and determination to follow through. They may require that we dare to look deeper at how we do things and question the status quo.

The number one takeaway from this chapter is that complexity due to quality issues and failure demand—bugs and errors—will create a loop that automatically increases pressure due to low control of the influx of work items, as illustrated by the image on page 69.

For people to dare question the status quo, we need a basic level of security—the knowledge that it's okay to question and improve things. Within team Cumulus, we had that security so we could start working on our complexity together. However, the external security, the trust between production, product management, and development had deteriorated and taken its toll over the years.

We needed to start collaborating more to gain trust in each other and get a mutual understanding of our challenges and goals.


Tactical decisions affect pressure
Shortcuts increase complexity
Strategic decisions affect complexity
Quality issues increase pressure
PRESSURE × COMPLEXITY
SECURITY

Remember

Complexity—conditions that impede us from efficiently completing a task—has two main categories:

- Cognitive load
- System inertia

Cognitive load can be affected by code complexity but also by non-code aspects such as:

- Context switching
- Product design
- Insufficient infrastructure and tooling
- Unpredictable influx of work

System inertia can be caused by:

- Constraining processes
- An abundance of roles with sole ownership
- Low flow optimization

CHAPTER 6

Team Cumulus: From Clutter to Clarity

Though the writing had been on the wall for some time, the departure of our development manager came as a shock. The bad vibes from the open conflict within management had naturally spilled over onto us, not that we took it out on each other, but as a team, we were dancing on the edge of disillusion.

The team desperately needed attention. Over the years, I had grown more interested in Lean software development and team building, so I volunteered to take care of the team and make sure development kept rolling.

Focus on Flow

The team had grown fairly rapidly out of necessity but also during a time when people were close to burnout. We were worn down from figuring out a reasonable way forward and keeping an unstable and badly aged system going, while at the same time onboarding new people.

The most acute indicator that things were bad was our sense of responsibility was ebbing away. Team members did not take the initiative the way they used to. Few volunteered to do peer reviews or pair programming when others needed help. I don't buy into the "fact" that a team cannot be larger than a particular size—I have seen large teams work well. However, in larger groups, you can hide in the numbers. You start thinking that someone else can do it. Those with school kids will recognize how, as soon as someone asks for volunteers in a parent meeting, everyone's gaze visibly loses focus and drifts into the distance. Well, that's what was happening in Cumulus.

There was reluctance to divide us into two separate teams, so we decided to split into two subgroups. Each subgroup would be responsible for specific "development tracks," which could be a new subsystem or part of the new interface. Each subgroup would also be responsible for helping and peer-reviewing each other. This clarified responsibilities and focus. We would also rotate support duties, reducing the need to sync urgent issues. The rest—retros, planning, architecture, after-work beer—we'd keep doing as one unit.

Each subgroup could have two or three different development tracks going, but we tried to avoid assigning tracks to any single individual. We would always try to have at least two people on everything. This helped ensure that if someone got stuck, they always had a buddy who was up to speed to bounce ideas off of. This would help us keep our momentum. We couldn't always achieve two per track, but we tried as much as we could. Like magic, the sense of responsibility returned.

Up to this point, because we had to prove our architecture would work in reality and at scale, a lot of new feature development had been done in a rapid quantity-over-quality manner. Now, we had to improve quality before pressing on, as production errors kept interrupting our flow. The cycle became three weeks of development followed by one week of stabilization and optimization. We would also prioritize development that made it possible to decommission legacy systems. Everything we could shut down saved us time and energy.

This improvement journey paid off. Our deployment pace increased from a three-month average of 8.5 days with production updates per month (which isn't bad!) to 17.6 days per month. This was an increase of more than 100 percent despite being down one developer since one of the guys had taken an extended parental leave.

Plan for Mutual Understanding

Our next move was to increase the involvement of our production experts in our workflow by introducing a product discovery process. Together, we tweaked it into what we came to call EvoDiscoDev—Evolution, Discovery, Development—which was a complete chain from idea to deployment without any "hard handovers." Developers got involved early, and production experts were involved all the way from design to implementation through workshops and by joining our dailies.

Not only did this increase our cross-competence collaboration, but it also provided us with a well-structured backlog and a great overview of what was going on.

Cumulus now consisted of eleven developers, two UXers, one domain expert, and one newly recruited development manager. A well-structured development process fed the backlog, and the number of urgent production issues was reduced to a point where the team could actually start doing proper Scrum.

I am not saying Scrum has to be the goal of an Agile transformation, but as anyone who has successfully implemented Scrum will tell you, it takes discipline and maturity to pull it off. With the whole product organization in much better shape, we had finally reached that level of discipline and maturity.

This was the point at which I started to wind down my engagement. I had been asked to take over coaching duties for another team and accepted the position.

The story of team Cumulus transpired over four years, and while it may sound like a smooth ride when described on so few pages, there were heated discussions and disagreements along the way. The price we had paid for lack of structure, mismanagement, conflict, and technical debt was stress. We rarely worked overtime, yet all of us were on the brink of burnout at one point or another.

The straw that nearly broke the camel's back was the conflict in management. There was constant bickering that, over time, turned into personal animosity and meetings filled with tension you could cut with a knife. It had become a hostile and toxic environment. Enabling a constructive and creative atmosphere with personal, cross-competence communication helped us get back on track. It created a better, more secure work situation for everyone. *And*, a better product!

CHAPTER 7

Increasing Security

A strong sense of security can make us brave enough to take on very challenging tasks, even if there is high pressure and complexity. If we know we're taking on the challenge as a team and that our manager has our back if things go south, we feel the security that keeps our stress level in balance.

People a lot smarter than me have said that necessity is the mother of invention. I think that's true in a lot of contexts, but not necessarily in the large modern corporate environment. In this setting, a more appropriate wording would be: "security is the mother of innovation."

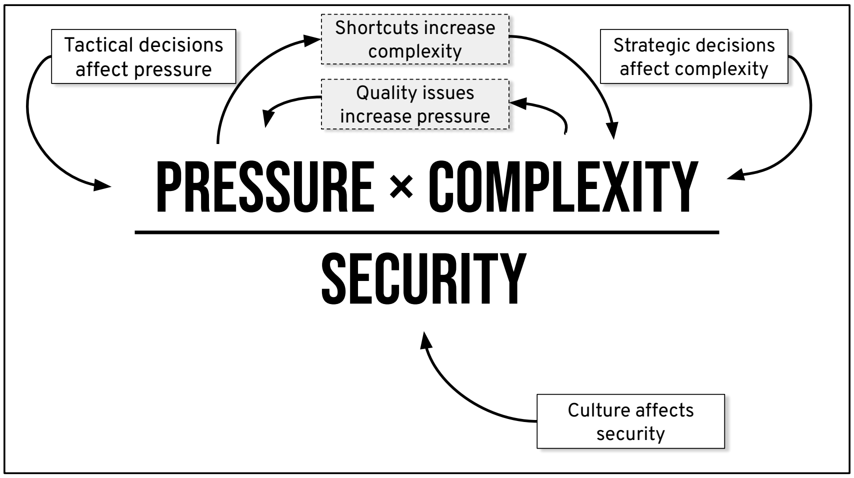
In this chapter, we will learn about what decreases and increases the feeling of security. It's security that tells us it's safe to dare to try new things and to speak our mind without risking losing prestige. If more people dare, more innovation and initiatives will come to fruition. If more people dare, more correct decisions will be made throughout the organization.

This sense of security largely comes down to the culture of a team, organization, or company.

Define "Culture"

Culture is a very complex thing, but the simplest way to describe it is how we speak to and about each other—the words we choose and the tone we use. Who has a say, and who doesn't? Do we tolerate gossip and backbiting? This does not catch everything related to a company's culture, but it's an indicator that carries a lot of impact.

The image on page 76 illustrates how tactics, strategy, and culture relate to the Pressure, Complexity, and Security variables of the Stress Equation.


Tactical decisions affect pressure
Shortcuts increase complexity
Quality issues increase pressure
Strategic decisions affect complexity
PRESSURE × COMPLEXITY
SECURITY
Culture affects security

Culture also reflects the values and behaviors we accept and award in promotions and paychecks, directly or indirectly. If you promote someone who is brilliantly competent but a nightmare to collaborate with, you are not only promoting the competence, you are also indirectly accepting and promoting the nightmarish collaboration skills. This sends the signal our culture not only values competence over collaboration but also rewards non-collaborative behavior.

I am not trying to downplay the importance of competence and mastery when trying to succeed in business. It's vital. We need driven experts to push boundaries and break new ground. However, it doesn't scale well unless there are a few drops of humility and empathy mixed in there. Big egos, who think work is a never-ending game of King of the Hill, are rarely a good basis for a secure company culture.

Appreciate Cooperation

Ron Westrum's research on company culture shows that high-performing organizations are characterized by cooperation, active information sharing, and shared risks and responsibilities.[1] While we need technical brilliance to excel, teamwork and collaboration are often the only ways we can accelerate. It's okay to seek prestige and individual recognition, but unless the organization gives collaboration the highest prestige, the ability to scale smoothly will be severely reduced.

1. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1765804/pdf/v013p0ii22.pdf

Let me rephrase that: no scale up is ever smooth; it's always challenging. But, if the ability to collaborate is a pre-existing challenge, scaling up will be extremely challenging.

Company culture is also about how we define great leadership. What is it that signifies and nurtures great leadership in our organization?

More than once, I have witnessed companies educate line and middle managers in psychological safety, coaching, and how to meet employees with empathy and understanding, only to then tolerate high-ranking managers bulldozing their way through meetings, openly shaming and yelling at people, and criticizing and disregarding recommendations from their own experts.

That kind of behavior can get very potent short-term results, but there is a disconnect between what's being taught and what's being practiced. It sends ripples of fear throughout the organization that will harm a company's culture in the long run.

Questions About Security

- What do we define as prestigious in our organization?
- If salary was defined by an algorithm, which parameters should this algorithm take into account? Which soft/personality skills and which hard/competence skills should the algorithm take into account?
- What nurtures and promotes great leadership in our organization?
- What constitutes great leadership for us?

Avoid Decreasing Security

I am the first one to admit that there is a serious mathematical flaw in the Stress Equation that we need to address—mainly to calm the engineers. Lack of security can itself be a major source of stress. When we are vulnerable to other people's whims and behavior, or if the company is doing badly and we fear losing our job, that is a stress multiplier.

This is not taken into account mathematically in the model, but the questions we need to ask ourselves are still captured in the security variable. Remember, the equation is not meant to be taken literally but to be used as a visualization for analysis and discussion.

The reason for using it as the denominator is to highlight the importance of raising security and think about what we can do to achieve that. This should be our focus because it is one of the most important aspects of lowering organizational stress levels.

Even so, it's probably a good idea to give some examples of what decreases security since looking for antipatterns is a good way to find places to improve.

Don't Send Mixed Messages

Clarity is one of those things communicated in leadership courses—clarity of vision, clarity of goals, clarity in expectations—but it's something that constantly proves hard to deliver. The importance of clarity also showed up in Google's study, Project Aristotle,[2] where clarity of goals, roles, and execution plans came in as the second-most important thing for high-performing teams.

In other words, we need clear expectations.

The most common mixed messages that I encounter are over ever-changing backlog priorities. Here's where I see some of the biggest misconceptions about business agility—that being agile means never having to make any final decisions.

If we never determine clear priorities and final commitments, managers can never make correct tactical decisions on where to focus. Even worse, from a stress perspective, managers can never rest assured that they made the right call. Likewise, ever-changing goals make it even harder to make the correct technical decisions when building software.

This is partly related to what was discussed in Focus On Flow Optimization and Structure the Work. If our goals are unclear or we try to achieve too many different goals, we can't optimize flow or structure our work properly. Likewise, with unrealistic amounts of parallel work, we negatively impact security.

When we try to make progress on too many things at once, we lose clarity on what's important. We may find ourselves acting like everything is more important than everything else, no matter how absurd that sounds. This spreads insecurity because we don't know what we should focus our efforts on when we get conflicting directives.

In the end, when we are constantly sent mixed messages, it's hard to feel confident in our leadership.

2. https://malcolmdrakes.wordpress.com/2023/05/15/building-effective-teams-lessons-from-googles-project-aristotle/

Without clarity in priority and commitments, there is a risk of debate fatigue, as we constantly need to fight for our cause. Or, it may lead to a reluctance to make any decisions at all, which potentially leads to more mixed messages. When everything is always up for negotiation or debate, the risk of clashes and conflicts increases.

Questions About Mixed Messages

- Are there situations where we wish our leadership could provide greater clarity?
- Are we collectively able to define and commit to common goals?
- Is there a consistency in the way we act and expect the organization to act?
- Are there established processes and policies that are not in line with the strategic goals and our cultural values?
- Are all leaders in the organization aligned and on board with the strategic goals?

Resolve Open and Ongoing Conflict

We sometimes talk about constructive conflict. I am a bit hesitant to call anything constructive a conflict. A mere disagreement or difference of opinion is not necessarily a conflict. I would label that as friction. Friction generates heat, and we shouldn't be scared of a little heat from time to time. Where there are passionate people, friction is bound to happen at some point.

It's when we leave things unresolved or pretend it's not there that we have a problem.

If left unresolved, that friction starts to build up and in the end becomes a major distraction. When the friction intensifies too much, we get stuck and are no longer able to move forward in a constructive way. That's when we have a conflict.

If unaddressed and untreated, this conflict can spiral into sheer hostility. This hostility is not always obvious, especially if you are new to a company or come in as a consultant. Its detectability depends on whether we have a hot or a cold conflict.[3]

3. https://hbr.org/2014/06/to-resolve-a-conflict-first-decide-is-it-hot-or-cold

Identify Hot and Cold Conflicts

A hot conflict is pretty obvious. Emotions run high, voices are raised, and people are often unable to stay objective and solution-oriented. It's an open tug of war.

This is naturally a stress factor for the parties involved, but also for people around them. We might start "walking on eggshells," afraid to bring up certain topics. Meetings become tense because everyone is anticipating "the eruption." We may feel forced to pick sides or fear being caught in the crossfire. In either case, it's a volatile and unpredictable situation.

On the other hand, cold conflicts can be harder to pinpoint. They can be the result of a hot conflict cooling off but still left unresolved. They could also be in the form of discord that has built up over time without any obvious outbursts of emotion. It's a situation where things go unsaid or are muttered after a meeting.

On a team level, cold conflicts can be a low-intensity stress factor for the parties involved, but they will likely spill over onto other team members as well. When there's a conflict, even a cold one, between people who work together on a daily basis, it *should* be detectable for any manager with personnel responsibility. It will harm the team's performance and collaboration. It affects the discussion climate in the group and their ability to take joint responsibility. There are numerous signs to look for.

If there is a cold conflict on a management level, it can be harder to spot. I sometimes compare it to a black hole—it may not be directly visible, but you can tell it's there by observing what's going on around it. For example, in its wake, there is usually misalignment or a lack of important organizational decisions. Cold conflicts can make it hard to get things done across departments. Personal contacts within the organization become important because there is a reluctance to solve issues or align on a higher level.

This might not be a direct stress factor for the conflicting parties because they might be pretty comfortable with the stalemate. However, the cost of misalignment and lack of cooperation is moved onto others—usually coordinating roles such as project managers. The problem is not immediately obvious, but project managers are left to deal with personal conflicts that they are neither prepared for nor responsible for resolving. They may start feeling anxiety before meetings where they know things will become problematic due to animosity between important decision-makers. Big projects that require cooperation and alignment across departments will suffer and potentially run aground.

This situation can easily become a toxic cocktail for anyone tasked with cross-organization projects.

Questions About Conflict

- In what situations do we tend to let disagreements go unresolved?
- Do we need to establish a decision protocol on how to resolve stalemates?
- Are there potential cold conflicts that are affecting our team negatively (external or internal)?
- Is there any conflict, hot or cold, that we wish was resolved once and for all?

Counteract Toxicity

With psychological safety[4] being widely discussed, there has also been some debate about toxicity and, in particular, toxic leadership. I would like to widen the perspective a bit, and also talk about situations that can become toxic, even if no one person's behavior is the direct cause.

There may be roles and individuals who get caught in the middle of organizational discrepancies, where their jobs become impossible because of how things are. These individuals are often under pressure from high complexity due to adverse circumstances and may have little or no support system. In fact, the manager who should be helping them solve the situation may be the same person who is pressuring them to deliver.

If these discrepancies are caused by unaddressed conflicts, there may be no one taking responsibility for solving the problem, leaving the individual to cope with the fallout.

Address Toxic Situations

In this situation, no one person is necessarily the toxic element, but the sum of its components becomes too much to bear. No one is directly to blame, but it's a case of burnout by inconsideration. Unrealistic expectations due to unclear goals, tight deadlines, and overcommitment in combination with low transparency, poor communication, and lack of collaboration—all of this together is a toxic cocktail for anyone tasked with driving larger projects forward.

4. https://hbr.org/2023/02/what-is-psychological-safety

I have seen this happen to project managers, line managers, product owners, release train engineers, integration leads, and so on. Usually, I see this with people in middle to lower management caught in the crossfire of conflicting interests in the organization. I am not saying it's exclusive to these roles, but drawing from my experience, they seem to be extra exposed to these circumstances.

When the individual finally succumbs to the stress, there is rarely a proper root cause analysis that views this as an organizational failure. Instead, it is viewed as regrettable, but "clearly" this person did not possess the tools to deal with the job—it's a case of the "wrong person in the wrong place." The organization is never to blame; the flaw lies with the burnt-out individual.

Seeing this happen one time too many is one of the reasons I am skeptical of organizational constructs that require an abundance of coordinating roles or very clear-cut areas of responsibility. It's the lazy leader's solution to a badly managed system.

Then, there is the matter of outright toxic people.

Avoid Toxic People

In leadership courses, there is often a message that no one person is ever the whole problem and this is something we need to remember to lead in a balanced and empathetic way. While true in most cases, I have had experiences where one person has indeed been the whole and entire problem. Leaders who manage and "motivate" through fear of public reprimands and repercussions are one example. Another is team members who lash out and blame everyone else for their own lack of dependability. If we are slow in responding to this type of behavior, we may find ourselves in a situation where the good ones leave and the bad ones linger.

I have had to deal with toxic behavior from high-ranking managers on a couple of occasions. Even when toxic leaders are in the minority, they somehow manage to take up the majority of space and energy. They are often blockers for real change since confronting them can be utterly exhausting. If they show signs of narcissism as well, they are highly unlikely to ever admit any fault of their own or change their minds. For your own protection, distance yourself.

Their manipulative behavior spreads waves of insecurity across organizations and causes a lot of damage. It's not always easy to put your finger on it, but some people just have a way of making you feel insecure or stupid—that it's your fault an argument has arisen. If you try to address their behavior, you are accused of being too sensitive, paranoid, or seeing things that aren't there.

It's not always a conscious strategy from the person manipulating you. It could be a behavior they have acquired because it produces the result they want—that they get their way. Nevertheless, intentional or not, it can cause widespread and severe damage.

I will not go deeper into the subject because the intention of this chapter is to focus on how to raise security and work on a better culture. However, we do need to be aware and watchful for toxic behavior on all levels of an organization.

Questions About Toxicity

I will not give any questions to discuss on the topic of toxicity and how to address it. This is a tricky subject, and since toxic people fall too far outside of my expertise, it would be irresponsible of me to pretend I have the answers on how to solve it.

If you are the victim of toxicity, I would contact an HR representative who can help out—failing that, distance yourself as fast as possible.

Enable Increase of Security

With toxicity out of the way, I think it's time to pick things up again and lighten the mood a bit. I believe the best way forward is to focus on what positively affects the level of security in teams and organizations.

Let me start by saying: I love engineers! My love of engineers is probably an extension of my love of nerds. People with deep knowledge always intrigue me, and I love to listen to them talk about their favorite subjects. Engineers often have a tendency toward nerdery!

Mind you, there is one big challenge for team building in engineering-heavy organizations. As soon as you can't explain something with logic or rational deduction, you enter la-la-land. Soft team-building exercises, where we talk about how we "feel about things," will swiftly produce an overwhelming amount of skepticism. If you start with that "kumbaya by the campfire crap," you will likely lose your audience in the blink of an eye. (I am generalizing, I know, but this has been true in the majority of cases—at least for me.)

Matters of psychological safety rarely come up spontaneously in most teams. It's an intriguing choice of words to say "hard skills" in reference to technical things and "soft skills" in reference to communication and self-organization when it's the soft stuff that often turns out to be the hardest. This reluctance to talk about soft issues in engineering contexts is actually one of the reasons for constructing the Stress Equation. In a setting where we want to stay rational and not talk about personal feelings, analyzing the system is a great way forward.

After a while, maybe we can start talking about the softer aspects of psychological safety and its importance to team performance.

Build on Psychological Safety

Google's Project Aristotle[5] catapulted "psychological safety" as a buzzword in the software world. In the unlikely event you haven't heard of Project Aristotle, it was a study conducted by Google to identify the key factors behind high-performing teams. Psychological safety came in at the very top. In fact, the top five were heavy in what could be labeled as soft, social factors. Dependability and meaningful work, for example, are more related to attitude and how we feel about our work than actual "hard competence."

With all that said, we should probably not neglect the aspect of technical competence when it comes to high performance in teams. After all, the study was conducted at Google, a company renowned for attracting some of the best tech minds of the past two decades. However, it does highlight the importance of social dynamics in helping a group reach a higher level of performance.

Those who have read the writings of Amy Edmondson, for example, *The Fearless Organization [Edm18]*, will not be surprised by this. In short, her studies show that teams with a habit of being open about ideas, concerns, and errors consistently performed better. This is when speaking your mind is not seen as challenging authority but as an attempt to further the performance of the team, and admitting one's errors is seen as a strength, not a weakness.

Building psychological safety takes time, and most teams need active help, coaching, and feedback along the way. In my experience, this element of time is often overlooked. Ultimately, psychological safety is about trust, and trust comes from healthy human relations. Healthy human relations can not be hurried. They can be helped along, but they can never be forced.

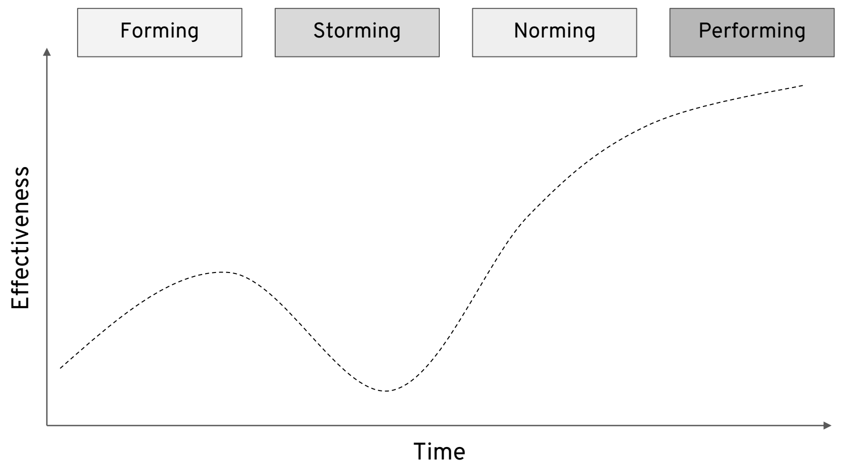
Nurture Stable Relationships

Susan Wheelan's research on group processes and team development shows that it takes many months for a team to reach the Performing stage in Bruce

5. https://malcolmdrakes.wordpress.com/2023/05/15/building-effective-teams-lessons-from-googles-project-aristotle/

Tuckman's model[6] of group development. Tuckman's model illustrates the stages groups go through as they develop and mature—Forming, Storming, Norming, and Performing. It is normally illustrated either as steps moving toward Performing or as a curve showing a team's performance as its maturity progresses over time.

The following image illustrates Tuckman's model as the performance progression of a team in relation to time.



While I think one should be careful in putting exact numbers on how long it takes, I have read averages ranging from six to twelve months. In *Creating Effective Teams [Whe10]*, Wheelan says it takes six months for a team to mature but that the number depends on how much time the group spends together. It then takes another few months for the team to reach the Performing stage of Tuckman's model.

However, very few teams reach that level of maturity unless they receive active help getting there. These averages are also assuming that the team remains stable—that there are no significant changes in personnel or organizational structures. It assumes that they have time to build relationships with each other and establish a way of working, both internally and with their external stakeholders and managers.

6. https://web.mit.edu/curhan/www/docs/Articles/15341_Readings/Group_Dynamics/Tuckman_1965_Developmental_sequence_in_small_groups.pdf

I have rarely worked with teams that remain completely stable in this way for six to twelve months. There is almost always significant change in one way or another, either due to personnel turnover, reorganizations, or scale-up.

Reduce Turnover and Reorganization

Turnover is a big challenge when trying to build high-performing teams, and my experience is that—from a performance perspective—it's more costly to add new members than to remove someone from the team. Given that the team still possesses the technical competence and domain knowledge to fulfill its mission, removing a member is less disruptive.

Constant company reorganizations are another folly that spreads change fatigue and greatly inhibits a team's susceptibility to coaching and internal improvement. If their surroundings keep changing, establishing an efficient way of working with external stakeholders and managers becomes next to impossible.

Scaling up is a more fun challenge, but a challenge nonetheless. Going into that would be a whole book in itself. Let's just say that giving teams the stability and support they need to become high-performing requires constant attention, patience, and proper strategy.

If an organization considers itself to be team-based, we need to be aware of this. Without it, we are relying on luck alone for creating psychological safety. A good place to start when trying to increase the level of safety and security is by showing trust through servant leadership.

Questions About Psychological Safety

- Do we have clear expectations placed on us? (Goals, roles, and plan of execution.)
- How do we actively work toward and ensure high psychological safety among our colleagues?
- How do we actively work toward and ensure high psychological safety among our leaders and managers?
- How do we react when someone admits a serious error?
- In what way do we make sure that the most junior member of our team is comfortable speaking their mind?

Practice Servant Leadership

There seems to be a tendency in the leadership development community to use the term “boss” as a derogatory term, almost the opposite of a leader. I am not entirely a fan of this trend.

There are times when someone has to take the responsibility of making uncomfortable decisions and the heat for doing so. On occasion, you need a boss to sort things out when there is misalignment. Just because there are bad bosses doesn’t mean all bosses are bad, and sometimes, there needs to be no question about who’s in charge.

A good boss is a good leader.

Appreciate the Managers

Likewise, I don’t like when the term “manager” is used as a synonym for a boring, bureaucratic function. Servant leadership is, by and large, about managing the stuff that needs managing so the operations of a business run smoothly. It’s an important job!

Good management is often invisible and thankless work. We only really think about it when it isn’t done properly, or managers get bogged down shuffling paper, and all their energy is consumed by keeping the status quo.

It’s not easy being a good manager! It takes thick skin and stamina, and I am very much an admirer of managers who fight corporate dragons on a daily basis. They are the unsung heroes of the company, taking heat from all directions when things get bad. I honestly don’t know how they do it! And then we demand they be playful, entertaining leaders, and inspiring communicators of visions as well.

Let’s get real about this, please!

Provide Support

To enable servant leadership, the trick is to make bureaucratic management as lightweight as possible so that the managers can focus on solving the issues and bottlenecks that hold you back from reaching your strategic goals. Scaling an enterprise in size always requires more structure, and the hardest part is finding the right balance.

Servant leadership is, at its essence, about bosses, managers, and leaders acting in the best interests of the employees and the organization by building bridges and resolving problems.

In short, their main job is to support, not to demand. Only when management is able to provide the needed support are we in a place where we can start putting demands on employees. Anything else is reckless.

As employees, we should feel that leadership is on our side and has our interests at heart, that we aren't micromanaged, and that we have autonomy in our workday and field of expertise. If there is a demanding circumstance where customers put extra pressure on us, we should feel that management is there to help us succeed and that they are keeping the road clear ahead of us, not chasing us with a whip.

Balance Employees and Customer Focus

With that said, there is another balance to be considered here between employee and customer focus. It's easy to fall into the habit of jumping as soon as a customer wants you to jump—to become solution-focused with customer problems. In contrast, there is often a focus on obstacles and costs when dealing with internal and employee problems.

If this is your reality, that you act as if you're living in serfdom under your customers, there is very little room for servant leadership. It becomes an empty phrase.

As a leader, you should always be on your employees' side. That is your job and duty. Sometimes, it's in the best interest of the employees to please the customer, but not always. Sometimes, the best way to please the customer is to please the employees, but not always.

I never said this was easy. If it was, everyone would get it right! For servant leadership to work, there has to be a level of trust in people and confidence that they will do the job to the best of their abilities. If they don't have the ability, there needs to be enough psychological safety to raise concerns and address them.

Questions About Servant Leadership

- Do we feel that leadership provides the conditions needed to do a good job?
- Are there situations when there is a tendency to micromanage?
- How do we act when we need to go against a customer's wishes?
- How can we build the confidence to say no to customers?
- How do we act when we can't please our employees?

Display Trust and Confidence

Trust and confidence are basic requirements for servant leadership. We need to know we can trust our colleagues to do what they have said they will do and be confident that they will do it to the best of their abilities. If a leader cannot feel trust and confidence in colleagues and coworkers, it's simply impossible to act as a servant leader, which is the foundation on which you build a high-performing organization.

Trust and confidence don't suddenly appear out of thin air. For someone else to be able to show trust and confidence in me, I need to show dependability and transparency—that I am dependable in my commitments, that I do the job well and don't just tick the boxes in a slapdash manner. I need to display transparency in what I am working on, my progress, and that I signal for help when things don't go according to plan.

Showing dependability and transparency is the most important reason for public burndowns, sprint scopes, and estimates in an Agile setting. These are tools to help teams learn to plan for focus, and to show what they will be working on and they can be trusted to do so. Far too often, I see burndowns and estimations promoted as a means to magically create predictability.

It *can* create a higher degree of predictability, but we should never confuse predictability with dependability.

Don't Confuse Predictability with Dependability

Predictability requires a stable system with a steady flow and a well-defined scope that does not fluctuate. Few organizations can provide their teams with that. In fact, the main reason for going Agile in the first place may have been to allow for late scope changes or to start work early on fuzzy requirements—learning as you go.

Those are all fine and valid reasons to go Agile, but you can't have that freedom and fuzziness of scope and still expect high-precision estimates and predictability. It's like trying to figure out the odds in a horse race without knowing which horses are going to be running or if they will be horses at all. You place a bet on what you think is a horse, and suddenly, there's a one-legged guy on a unicycle wearing the number you've bet on. And then your manager gets annoyed that you placed such a stupid bet.

Over time, I have grown very weary of estimates. There is usually a demand for estimates far too early in the process. Estimating poorly defined work packages, which are still open to significant scope change, is a waste of time. Furthermore, there appears to be a clear correlation between estimates turning into promises and the distance they travel: the farther an estimate travels from the team, the more of a promise it turns into.

Don’t Negotiate on Estimates

This tendency of estimates turning into promises is a major source of conflict and distrust. It may even sow doubt in the whole concept of Agile. Companies turned to Agile for promises of predictability by introducing story points and velocity measurements, and then it turns out it wasn’t quite that simple.

I have even been subjected to estimation haggling where customers and stakeholders call the team’s estimations into question and want them reduced because they exceed the effort they were expecting.

There are a whole host of things that need to be in place for even a remote shot at predictability—things that may contradict other capabilities that you want to keep.

Don’t use predictability as your basis for building trust and confidence. Build trust through joint scope breakdowns, realistic planning, and public sprint demos. In the end, the absolute best way of establishing trust and building human relations is face-to-face communication. Developing healthy human relations is the only way to build psychological safety. There is simply no way around that.

Collaboration over estimation haggling, as it were.

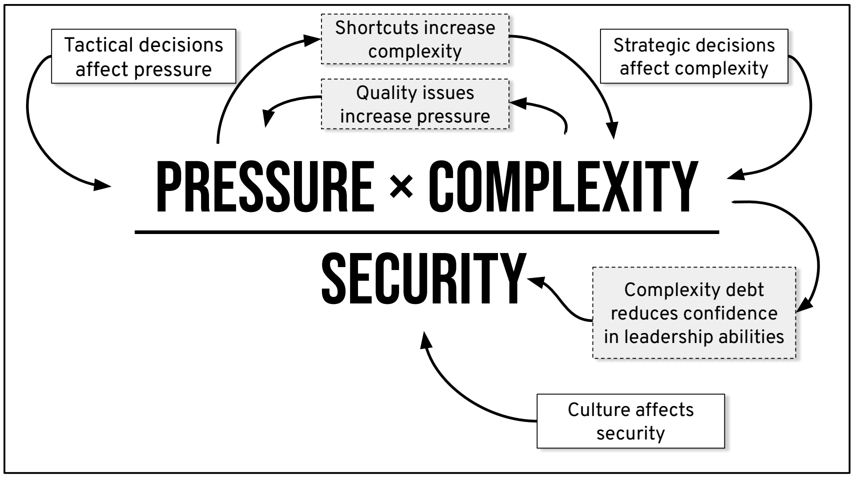
Questions About Trust and Confidence

- What are we doing today that makes us feel trust and confidence in each other?
- What could we do to increase the feeling of trust and confidence in each other?
- Is there anything in the behavior of our leadership that signals distrust in our work and efforts?
- In what way is our team showing dependability and transparency?
- What could we do to increase our dependability?
- What could we do to increase our transparency?
- Is there something that we are working on that we are not fully transparent about? If so, why?

Wrap Up

A culture of security and trust takes a long time to build but can quickly be ruined and razed to the ground by toxic elements. In the end, trust and safety largely come down to sound and stable human relations and clear expectations. We must get to know each other, working and experiencing things together, as we strive towards a common goal. This is why collaboration and direct communication are such important things.

There is one thing I want to emphasize a little more regarding complexity in relation to security. If we keep doing business on complexity credits, with ever-growing cognitive load and system inertia, employees will likely experience a reduced trust in the leadership's capabilities. Reduced trust in leadership will impact the sense of security negatively, as illustrated by the following image:


Tactical decisions affect pressure
Shortcuts increase complexity
Strategic decisions affect complexity
Quality issues increase pressure
PRESSURE × COMPLEXITY
SECURITY
Complexity debt reduces confidence in leadership abilities
Culture affects security

Throughout the book, I have often talked about communication and collaboration, because they helps us on so many levels. Stress tends to grow and fester when we feel alone. Just talking about it can be an important outlet—using communication and collaboration to solve it is essential.

So, let's take a look at how I use the Stress Equation to facilitate workshops and group discussions on the subject.

Remember

We should focus on what we can actively do to increase Security, but we need to be aware of what decreases it:

- Mixed messages
- Open and ongoing conflict
- Toxic circumstances or toxic people

We can actively increase Security in our colleagues by:

- Acting to increase psychological safety
- Practicing servant leadership
- Showing trust and confidence in those we work with

CHAPTER 8

Using the Stress Equation

Now that we better understand how and where stress can emerge on a systemic level, it's time to try improving things. Naturally, the best way to understand how to improve is to involve those affected by potential changes. The Stress Equation was designed to be a workshop concept and a model for continuous improvement.

In this chapter, I'll detail how I have used this method to create backlogs for transformation and leadership teams.

Prior to using the equation for improvement workshops, I meet with the department's management team to discuss which questions they think are the most important to focus on.

This book provides a whole battery of questions, but for an initial workshop, it's good to start on the more general ones, like "What creates unnecessary pressure on our team?" or "What should be easy for us to do but is cumbersome?"

This opens things up for a wider dialogue, which is exactly what we want to get going. We usually decide on a couple of questions per variable, where we can be more specific—especially if a topic has already been identified as needing improvement.

To start each workshop, I give a half-hour presentation on the Stress Equation. I explain the concept and give examples of what affects the variables—much like I have done in this book but more condensed. This gets everyone in the same frame of mind about how to look at the problem and provides a common reference for the discussions.

After the presentation, we have a short Q&A to answer any questions.

Discuss in Smaller Groups

For discussions, we break out into smaller groups—if possible, team by team, as people tend to feel more secure in that setting. We ask someone in each group to take notes, preferably someone with readable handwriting, if we want to collect the notes afterward. Be open about wanting to collect the notes for summarization so everyone is aware and prepared for it!

It's important to point out that the groups should distinguish between things within their sphere of influence vs. what is not. Things they can influence themselves should be topics for their team retrospectives and can be solved by each team.

The groups are given about 90 minutes to talk. If we only have an hour for discussions, we limit the number of questions to one per variable. I recommend doing two questions per variable, if possible, because it tends to spark more discussions.

Two hours is usually a bit of a stretch. People will lose steam—unless there is coffee and cake on offer while talking.

Reconvene and Gather Thoughts

When time is up, everyone gathers again. We review the questions, and each group is encouraged to share what they have been discussing. When things have been discussed beforehand in a smaller group, people are often more open to talking in a bigger group. This is an important effect that we want to achieve: the openness to talk about stress.

How we do this depends on how much time we have and the total size of the group.

If we only have a short time or a large group, we just use this as a venting session. We ask permission to collect the notes in case there are sensitive topics people are uncomfortable talking about in a big room. After the workshop, notes are summarized—keeping the input anonymous—to give us an aggregate view. From that aggregated view, we can identify common areas that need improvement.

If we have more time and a manageable number of people, I ask each group to present their topics and put them on sticky notes on a whiteboard. We try to group similar topics as they are presented. When all discussion groups have presented their notes, everyone gets to dot-vote[1] on which topics (or group of topics) they think are most important.

During the remaining time, we discuss the top-voted topics in deeper detail, hoping to find actionable ways to improve the situation.

Management then uses the result for strategic discussions and as input for continuous improvement or a transformation team. It's important that the result is actually used to make improvements. If nothing happens, people will be disappointed and feel less confident in the company's ability to make improvements.

Collaborate in Retrospectives

All of this said, you don't need to arrange a big workshop to use the equation. The questions provided throughout the book can be used on their own to spark discussions in team retrospectives or within leadership.

I spoke to a release train engineer in a SAFe organization, who had started asking questions about unnecessary pressure and what had been cumbersome after each product increment. This helped his team to make small improvements.

It's essential to make positive change easy. Small, continuous steps can make a big difference over the course of a year. As long as we have ways to talk about it and find easy, actionable steps in the right direction, things will improve. It was through constant evaluation and tweaking that team Cumulus got better control of the influx of work items.

Wrap Up

At the time of writing, this chapter illustrates one of the ways I use the Stress Equation, but I will probably keep revising and changing as time passes. As I mentioned earlier, I am something of a method agnostic and believe things need to be tweaked for the context in which it is being applied.

Before wrapping up the book, I think it's a good idea to take a look at the product discovery process that Cumulus put in place to increase collaboration and get a better overview of new development.

1. https://en.wikipedia.org/wiki/Dot-voting

CHAPTER 9

Collaborating on Product Discovery

Security may have made Cumulus persevere, but I maintain that better backlog management and a higher focus on quality were two of the key factors that finally dug us out of the technical hole that we were in. Since my time with Cumulus, I've worked with many more organizations and teams, and poor backlog and portfolio management are almost always a central part of their challenges.

In this chapter, you will learn what EvoDiscoDev looked like and how it was used. EvoDiscoDev was tweaked for about two years before reaching the form I am about to describe, and I am sure it kept evolving after I left. So, this will be a snapshot in time, not necessarily a final destination. However, I think it's important to give an idea of what a functioning portfolio and backlog management look like. It may not be right for your context, but it could inspire you to try something different.

Make It Visual

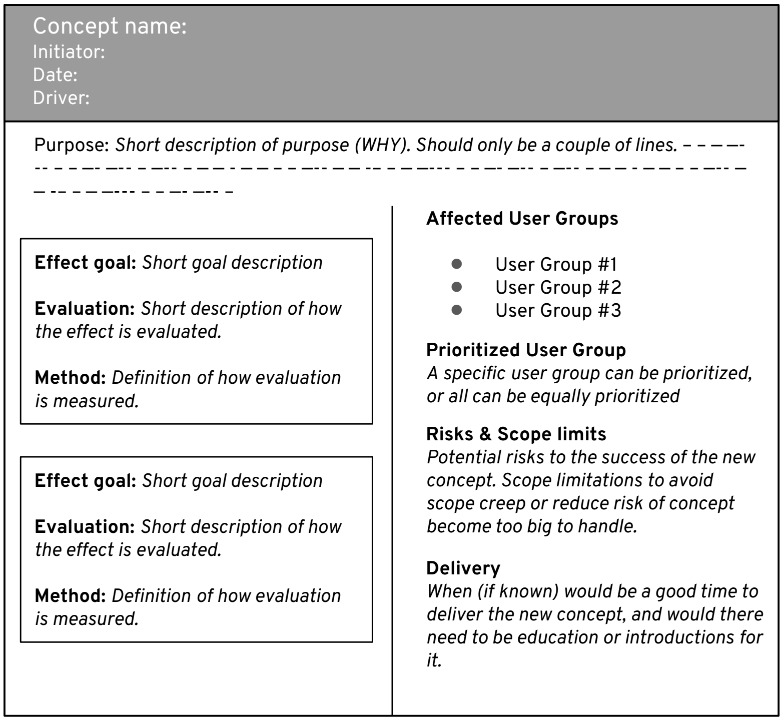
Part of the challenge in designing a coherent development process is visualization—something that gives you a quick overview of status while providing easy access to details if needed. Few things beat Kanban boards for this purpose. Sadly, many companies have settled for Jira's very limited standard visualizations. Jira is a great ticketing system, and a well-configured implementation is fantastically powerful. But, much like Jira itself, most configurations end up heavily bloated to the point where you can't see the forest for the trees.

So, for this demonstration, I am going to stick to customized Kanban boards. With powerful visualization tools like Miro or Mural, there really is no excuse not to create better boards—even if you work predominantly remotely.

Manage the Portfolio

The starting point in EvoDiscoDev is a *concept*. A concept is a brief description of what we want to achieve. These are usually fairly large undertakings. We are talking weeks of development time, possibly months. At this point, however, we won't take this time into consideration. We don't try to estimate it. Right at the beginning, a concept should be viewed as a wish, not a placed order.

The following image shows the standard template used for concepts.



Concept name:
Initiator:
Date:
Driver:

Purpose: *Short description of purpose (WHY). Should only be a couple of lines.* – – – –-
-- – – –- –-- – –-- – – – - – – – – –-- – – -- – – –--- – – –- –-- – –-- – – – - – – – – –-- –
– -- – – –--- – – –- –-- –

Effect goal: *Short goal description*

Evaluation: *Short description of how the effect is evaluated.*

Method: *Definition of how evaluation is measured.*

Effect goal: *Short goal description*

Evaluation: *Short description of how the effect is evaluated.*

Method: *Definition of how evaluation is measured.*

Affected User Groups

- User Group #1
- User Group #2
- User Group #3

Prioritized User Group
A specific user group can be prioritized, or all can be equally prioritized

Risks & Scope limits
Potential risks to the success of the new concept. Scope limitations to avoid scope creep or reduce risk of concept become too big to handle.

Delivery
When (if known) would be a good time to deliver the new concept, and would there need to be education or introductions for it.

A concept would, of course, have a name for identification. An initiator was specified to keep track of who actually thought this was a good idea in the first place—in case clarifications were needed.

We kept track of the date the suggestion was made so we'd know whether the idea could be scrapped or not. A two-year-old concept was probably either irrelevant or never going to get done. So why keep it?

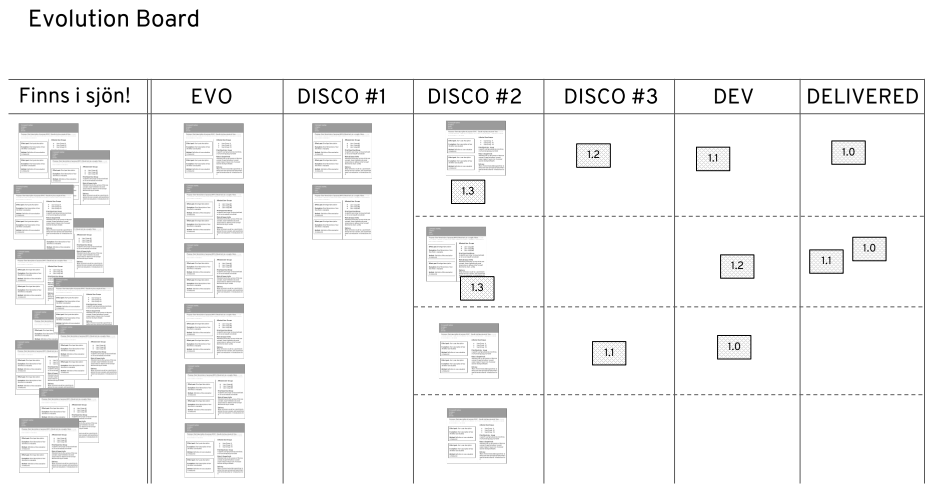
The purpose of the concept was a short description of why we needed this. It should be kept short and to the point.

Finally, we described the outcome in one or more effect goals—how would we be able to tell whether this concept was working or not?

In the first iteration of filling out the concept, all the rest of the fields were left blank, including evaluation and method for checking the effect goals. There was no need to spend more time on this until we decided to investigate it further.

The *Evolution* group was responsible for prioritizing and keeping track of concepts. This was essentially product management, with representatives from production and software. They'd keep a Kanban board visualizing the priority and status of each concept.

The following image illustrates the Evolution board:



The EVO column was a prioritized list of concepts that would be put into the *Discovery* phase when we had the capacity to take it on.

The initial prioritization was made on a "gut feel" from the Evolution group, based on how many users would be interested in this concept and how valuable or important it would be for them. So, if something was thought useful, it would get higher priority. If the concept was only useful to a few users but very important, it would get higher priority. If something was thought to be important to a lot of users, it would consequently get a very high priority. We did not estimate the effort at this time.

The DISCO columns showed the different stages of discovery that each concept was currently in.

One of the artifacts of DISCO #2 was a feature map of how we would slice and deliver the concept. If a concept took months to deliver in full, we wanted to know the checkpoints where we could push a minimum viable update to production (even if only to select users) to get feedback on what we had produced.

These feature slices provided the basis for what kept moving down the Kanban board. No numbered stickies? The concept stays in DISCO #2.

This way, we had the option to cancel further development if things were not working out as expected or if the effect goals were successfully met without further development. The last slices of each feature map were mainly nice-to-haves, rather than must-haves and could sometimes be skipped to get on with more pressing deliveries.

Discover the Way Forward

When a concept was put into DISCO #1, a driver was assigned. The driver was responsible for booking and facilitating the Discovery group's meetings. Each concept had a Discovery group of its own. The first meeting of such a DISCO group did not always include developers but always included a representative from the Evolution group, a UXer, and one or two production experts who would act as user representatives. Quite often, the concept needed to be broken down and specified further before it was meaningful to involve developers; however, if their input was requested, someone would join.

The breakdown of the concept was usually made with classic User Stories.

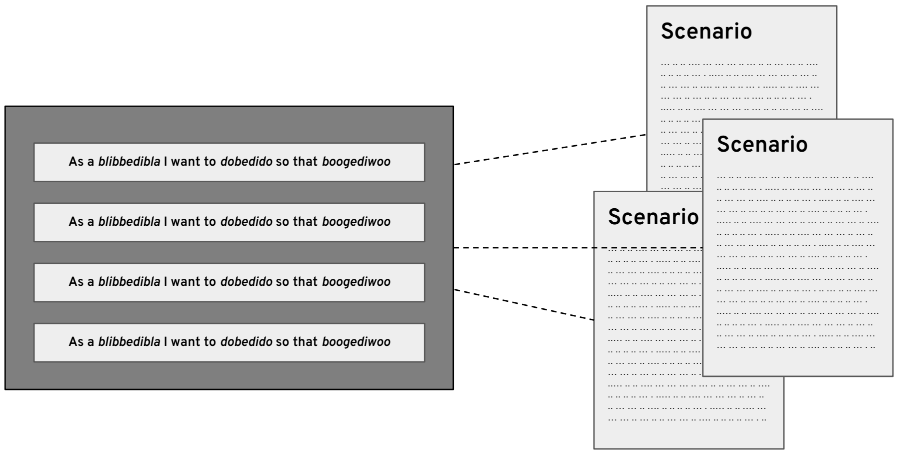
> As a … I want to … So that …

If deemed necessary, as illustrated by the image on page 101, more detailed, one-page (often half-page) scenario descriptions would be written as input for DISCO #2.

This breakdown provided better insight into which roles or competencies would be affected by the change and kept in the loop about the update. Sometimes, we created a rudimentary mock-up of a proposed user interface.

After a quick sync with the Evolution group to check that things were moving in the expected direction, the concept moved into DISCO #2 status. At this point, at the latest, two developers would be added to the DISCO group.

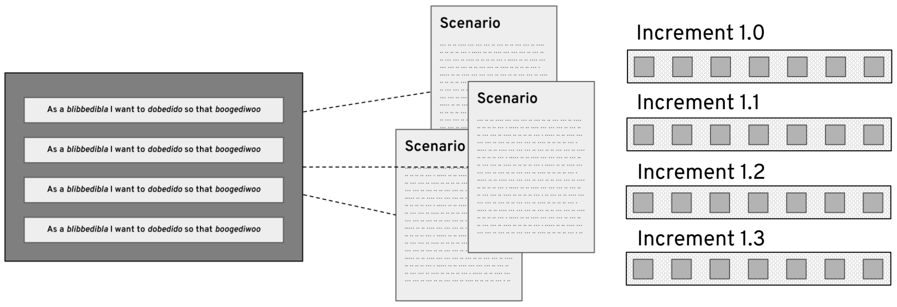
The main objective for the DISCO #2 phase was to provide a feature map of how the concept would be realized, with a rough idea of what would be included in each increment. This would sometimes require a couple of iterations, as technical investigations and architectural decisions might be needed


As a *blibbedibla* I want to *dobedido* so that *boogediwoo*
As a *blibbedibla* I want to *dobedido* so that *boogediwoo*
As a *blibbedibla* I want to *dobedido* so that *boogediwoo*
As a *blibbedibla* I want to *dobedido* so that *boogediwoo*
Scenario
Scenario
Scenario

before moving forward. Each increment was represented by a numbered sticky on the Evolution board.

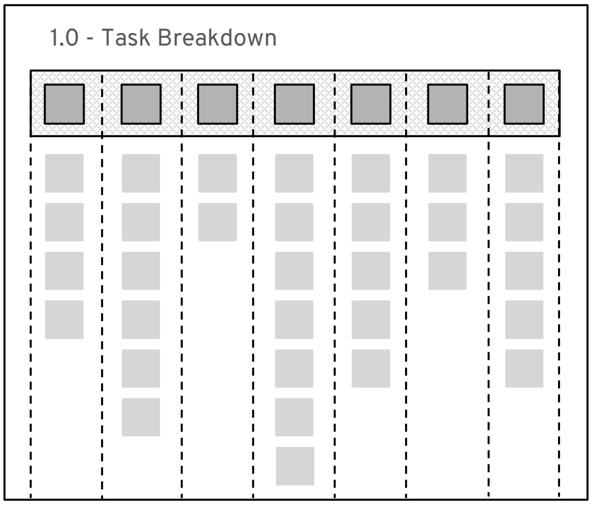
Risks were identified and evaluated when this was done, and scope limits were added to avoid potential scope creep. By clearly defining what would *not* be delivered, we avoided a lot of debates during development and learned the habit of questioning late scope changes.

In the following image, the stickies in each column represent the steps or tasks needed to complete the delivery. These stickies were not displayed on the Evolution board but kept as input for DISCO #3 and DEV.


As a *blibbedibla* I want to *dobedido* so that *boogediwoo*
As a *blibbedibla* I want to *dobedido* so that *boogediwoo*
As a *blibbedibla* I want to *dobedido* so that *boogediwoo*
As a *blibbedibla* I want to *dobedido* so that *boogediwoo*
Scenario
Scenario
Scenario
Increment 1.0
Increment 1.1
Increment 1.2
Increment 1.3

When the feature map was finished, a ballpark estimate could be done for each increment. This was synced back to the Evolution group for a decision to move into DISCO #3—the final preparation step before active development.

In DISCO #3, the UXer drew detailed images of the user interface. Since we worked with Trello, the intended developers would break the increment scope down in a separate board into more bite-size activities—subtasks to speak Jira.

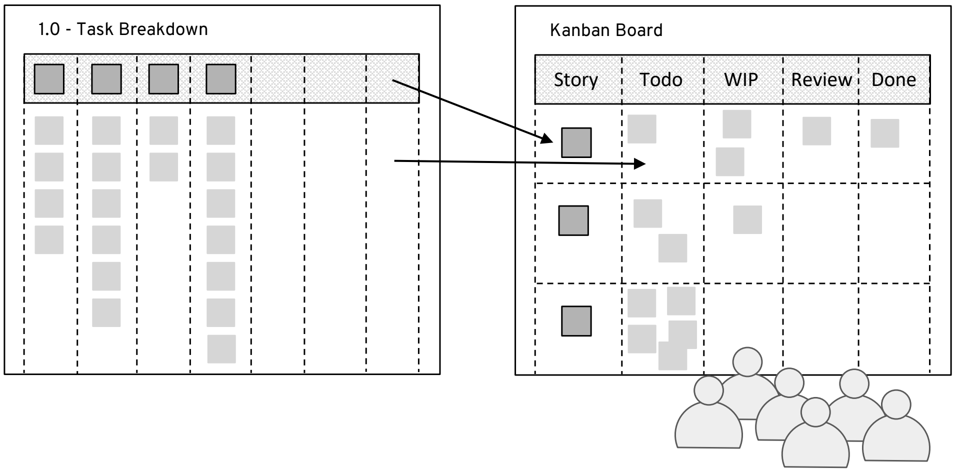


The DISCO steps would normally take place over several weeks, spending an hour or two at a time instead of doing everything in one go. This was a strength of the process since group members often came up with good ideas in between workshops. This is something I have come to realize is a benefit of a slower “maturation” process. Ideas and thoughts often go on in the back of our heads and may pop up at random moments. This is much like how developers working on a code problem realize the solution at random moments when doing something completely different—like brushing their teeth before going to bed.

After DISCO #3, things were prepared for development.

Get It Developed

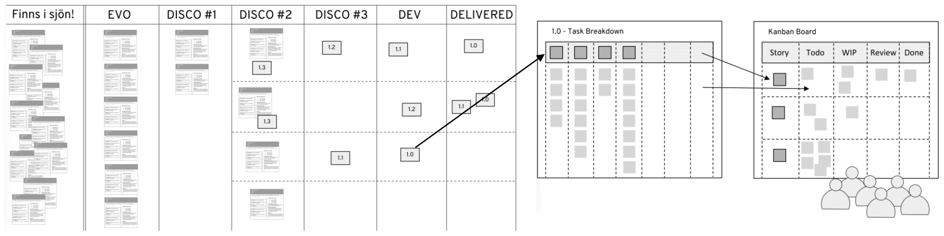
Once a concept was in active development, it was simply a matter of moving the different tasks to the team’s Kanban board and having at it, as illustrated by the image on page 103. Naturally, more work was discovered when the coding started, but the planning helped with focus and scope creep prevention. DISCO #3 was not intended to provide an exact work log but as a focus for development and to warn us when work started to grow unexpectedly.



Representatives from the responsible Discovery group were expected to show up at daily stand-ups to answer questions and comment on development. They didn't always make it daily, but they showed up at least a couple of times per week.

The team's Kanban board was, in reality, not quite as clean as depicted since there were still production emergencies needing room and attention on the board. But to illustrate the concept, I decided to keep the clutter down.

The process and visualization, as illustrated in the following image, provided a very easy overview of status and progress:



Even though the Evolution group focused on their board, they periodically asked for a more detailed status update. All we had to do was show them the corresponding Trello boards for each concept in development. There was also a standing invitation to all demos.

The new process eased communication, as we now had a common language and structure with expected steps and checkpoints. It's interesting how a lack of communication often causes inefficiencies and widespread stress as a result.

Wrap Up

This chapter was a summary of the EvoDiscoDev process. It had more details, like workshop concepts on how we tried out mock-ups of user interfaces, but the point is to give a quick overview of what a collaborative process looks like. This was our way of creating a common language and structure for ease of collaboration.

This whole book is an attempt to create a common language and structure to ease communication, but the subject is stress rather than product discovery. One of my main goals was to keep things brief and to the point, so because we're all strapped for time, it's probably time to wrap things up.

CHAPTER 10

Increasing Performance, Not Pressure

How well do you cope with stress?

It's a strange question to ask in a job interview because it's really just a rephrasing of the question: *How well do you think you will cope with this company's inability to balance pressure, complexity, and security?*

The Stress Equation is my attempt at explaining the technical, organizational, and cultural challenges I often encounter in software departments and companies.

When writing a book like this, it's easy to find yourself on a high horse, pointing out what everyone else is doing wrong and what they should be doing instead. I have tried to keep a nonjudgmental tone, even if a tad bit sarcastic from time to time because the fact of the matter is that creating and maintaining a balanced work environment is really hard—especially when trying to be ambitious at the same time, and I do think we should aim at being ambitious.

However, don't let unreasonableness masquerade as ambitiousness, like when we forget that prioritization is not about making a wish list. Prioritization is about removing and putting less important things out of our minds, even if we *wish* we could do them all!

Through bad prioritization and workflow control, we let unreasonable pressure increase, and that is when we start to take shortcuts and complexity credits. Over time, this becomes a costly practice, both moneywise and in our ability to excel when it really matters—when innovation and high pace are of the essence.

It's easy to point at others when things are not working well, and even though management and leadership need to be committed to creating a good place to work, we need to realize that everyone in an organization is responsible for keeping the Stress Equation in balance.

This is especially true with the Security variable.

A big part of this is to be mindful of the people around us and to help each other solve the challenges we face. It becomes much easier to love one's work when the equation is in balance, and people who love their work do a much better job.

I hope this book has given you a new perspective and, by doing so, an incentive to act. By getting the dialogue and improvement going, you can increase performance without added pressure while at the same time decreasing the stress in your organization and getting more work satisfaction.

With a bit of luck and determination, we might in the future hear interview questions like: *How well do you cope with a relaxed but high-performing work environment?*

But until then, we likely have a few things to improve!

Bibliography

[Edm18] Amy Edmundson. *The Fearless Organization*. John Wiley & Sons, New York, NY, 2018.

[Gol90] Eliyahu Goldratt. *Theory of Constraints*. North River Press, Great Barrington, MA, 1990.

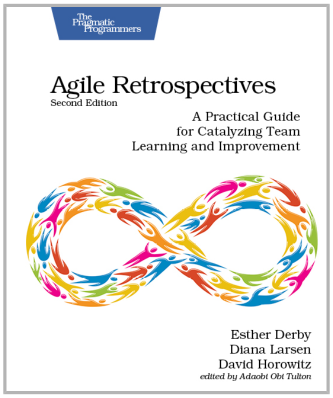
[MT14] Yves Morieux and Peter Tollman. *Six Simple Rules: How to Manage Complexity Without Getting Complicated*. Harvard Business Review Press, Boston, MA, 2014.

[Nor11] Alfred North Whitehead. *An Introduction to Mathematics*. Cambridge University Press, Cambridge, United Kingdom, Digital Collection, 1911.

[Sen94] Peter M. Senge. *The Fifth Discipline: The Art & Practice of the Learning Organization* . Doubleday, New York, NY, 1994.

[Whe10] Susan Wheelan. *Att Skapa Effektiva Team.* Studentliteratur AB, Sweden, First Edition (Swedish), 2010.

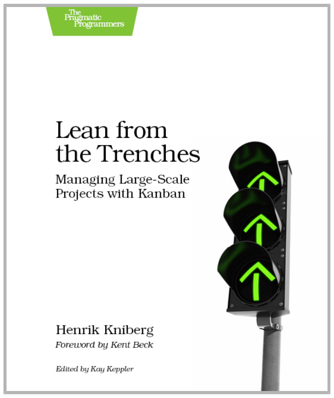
Thank you!


The Pragmatic Programmers

Agile Retrospectives
Second Edition

A Practical Guide
for Catalyzing Team
Learning and Improvement

Esther Derby
Diana Larsen
David Horowitz
edited by Adaobi Obi Tulton


The Pragmatic Programmers
Lean from the Trenches
Managing Large-Scale Projects with Kanban
Henrik Kniberg
Foreword by Kent Beck
Edited by Kay Keppler