

Praise for

Making Work Visible

“I love this book—Dominica DeGrandis talks about the chronic problems we all have in knowledge work and technology work in a way that is breezy, familiar, and often irreverent, but also shows off decades of learnings and concrete techniques we can quickly adopt, both at work and at home. Also wonderfully rewarding is when DeGrandis describes the theory of why these practices work, in a way that is accessible and enlightening.”

—**Gene Kim**, author, researcher, and founder of IT Revolution

“*Making Work Visible* is the quintessential book on visualizing work and understanding work management dysfunctions. Dominica has done brilliant work bringing these dysfunctions to life as the Five Time Thieves, and showing how these thieves rob your capacity and effectiveness. We use *Making Work Visible* at our company to build a common understanding of how vital it is to visualize and manage work effectively. I’m so excited to see this work expanded in the 2nd edition.”

—**Scott Prugh**, CTO, CSG International

“*Making Work Visible* is a must-read for everyone involved in digital transformation. The foundations outlined are critical to understand at the individual, the team, and the organizational level. If you haven’t yet, make *Making Work Visible* the guiding philosophy of your transformation—and start today!”

—**Mik Kersten**, CEO of Tasktop and author of *Project to Product*

“It is about time someone addresses time theft (aka the perfect crime) head-on. Not only does Dominica provide a lot of the why behind the forces that cause us to make bad decisions about our time, she also provides ideas of what to do about them. I wish I had this book when I took my first management job!”

—**Julia Wester**, Lean Consultant and Blogger at EverydayKanban.com

“*Making Work Visible* is a conversation we need to have now more than ever, regardless of whether it is about transformational change or governance, risk, and compliance.”

—**John Willis**, Sr. Director Global Transformation Office, Red Hat

“The most practical book I’ve seen on making processes lean. Dominica’s deep experience coaching companies is fully on display as she walks the reader through a series of exercises to find waste and eliminate it—or, in her terms, to catch those sneaky ‘time thieves’ in the act. Read this on a Sunday and you’ll want to start trying out the exercises on Monday!”

—**Mark Schwartz**, former CIO of US Citizenship and Immigration Services and author of *The Art of Business Value* and *A Seat at the Table*

“*Making Work Visible* makes work livable. It is a practical, actionable guide for organizations stifled by too much work-in-progress and the frequent interruptions that delay delivering value to customers and fuel employee burnout. *Making Work Visible* belongs on the shelf of every high-performing organization that respects their talents’ time and capacity.”

—**Aimee Bechtle**, Head of Solutions Architect, Mid-Atlantic and Carolinas Enterprise, Amazon Web Services

“Many of us wear our busyness as a badge of honor. In *Making Work Visible*, Dominica DeGrandis shows us how we can make hidden work-in-progress visible, to clearly see the effect it has on our ability to get things done. Once we can see it, she dives deep into the hidden aspects of our WIP that steal our time, energy, and productivity, along with strategies for combating each of them. *Making Work Visible* helps us to take a step back from all that busyness and really see.”

—**Chris Hefley**, Chief Revenue Officer, Retrium

2nd Edition

Making Work Visible

2nd Edition

Making Work Visible

**EXPOSING
TIME THEFT TO
OPTIMIZE
WORK & FLOW**

DOMINICA DeGRANDIS
FOREWORD BY TONIANNE DeMARIA

IT Revolution
Portland, Oregon



25 NW 23rd Pl, Suite 6314
Portland, OR 97210

Second Edition Copyright © 2022 by Dominica DeGrandis
First Edition Copyright © 2017 by Dominica DeGrandis

All rights reserved. For information about permission to
reproduce selections from this book, write to
Permissions, IT Revolution Press, LLC, 25 NW 23rd Pl, Suite 6314,
Portland, OR 97210

Second Edition
Printed in the United States of America
26 25 24 23 22 1 2 3 4 5 6 7 8

Cover design by Belinda Bowling, Joy Stauber,
and Richard Weaver, Stauber Brand Studio
Interior design by Devon Smith
Cover and interior illustrations by Dominica DeGrandis
Author photograph by Laurence G. Cohen

Library of Congress Control Number: 2021953422

ISBN: 9781950508495
ePub ISBN: 9781950508501
Web PDF ISBN: 9781950508518
Audio Download: 9781950508525

For information about special discounts for bulk purchases or for
information on booking authors for an event, please visit our website
at www.ITRevolution.com.

MAKING WORK VISIBLE, SECOND EDITION

I dedicate this book to my greatest inspirers:
my four brilliant children, Rachel, Robert, Angelo, and Augustus.

You teach me more about life and joy than any career
accomplishment possibly could.

CONTENTS

Preface to the Second Edition	xv
Foreword	xxiii
Introduction: Work and Flow	xxvii

PART 1 THE FIVE THIEVES OF TIME 1

1.1 Too Much Work-in-Progress (WIP)	3
1.2 Unknown Dependencies	13
1.3 Unplanned Work	21
1.4 Conflicting Priorities	25
1.5 Neglected Work	31

PART 2 HOW TO EXPOSE TIME THEFT TO OPTIMIZE WORKFLOW 37

2.1 Make Work Visible	41
<i>Exercise: Demand Analysis</i>	51
<i>Exercise: Identify Work Item Types/Categories</i>	52
<i>Exercise: Card Design</i>	53
<i>Exercise: Workflow Mapping</i>	53
<i>New Exercise: Making Work Visible with a Value Stream Canvas</i>	54
2.2 Ambush the Ringleader	59
<i>Exercise: Explore the Five Reasons We Take On More WIP</i>	65
<i>New Exercise: Setting WIP Limits</i>	66
2.3 Expose Dependencies	73
<i>New Exercise: Dependency Matrix</i>	80
2.4 Committing the Perfect Crime—Unplanned Work	85
<i>Exercise: The Interruption Reduction Experiment</i>	91
<i>New Exercise: Unplanned vs. Planned Work</i>	92
2.5 Prioritize, Prioritize, Prioritize	95

<i>Exercise: Visualize Priorities</i>	103
2.6 Preventing Negligence	105
<i>Exercise: Create an Aging Report</i>	110
<i>New Exercise: Conflicting Prioitization</i>	111
2.7 Useful Board Design Examples	115

PART 3 METRICS, FEEDBACK, AND CIRCUMSTANCES

127

3.1 Your Metrics or Your Money	129
3.2 The Time Thief O'Gram	143
3.3 Operations Review	147
3.4 The Art of the Meeting	153
3.5 Beastly Practices	161
Conclusion: Calibration	171
Afterword to the Second Edition	179
Glossary	187
Notes	195
Index	201
Acknowledgments	213
About the Author	217

FIGURES & TABLES

Note to the Second Edition

Table 1. Prepare for Objections with Key Metrics xviii

Introduction: Work and Flow

Figure 1. Builds Don't Take That Long xxxi

PART 1 THE FIVE THIEVES OF TIME

1.1 Too Much Work-in-Progress (WIP)

Figure 2. Prep Implement Feedback Board 10

1.2 Unknown Dependencies

Figure 3. Three Dependency Chart 17

PART 2 HOW TO EXPOSE TIME THEFT TO OPTIMIZE WORKFLOW

2.1 Make Work Visible

Figure 4. Visibility Grid 41

Figure 5. The To Do, Doing, Done Board 43

Figure 6. Balanced Work Item Types 47

Figure 7. Work Item Type Example 49

Figure 8. To Do, Doing, Done Board with Colors 50

Figure 9. Expanded Doing Column 50

Figure 10. Production Incident Workflow Example 56

Figure 11. Request Feature Workflow Example 56

2.2 Ambush the Ringleader

Figure 12. Expose WIP 61

2.3 Expose Dependencies

Figure 13. Physical Dependency Matrix 75

Figure 14. Arts & Crafts Dependency Board 76

Figure 15. Dependency Swimlane Board 76

<i>Figure 16. Dependency Tags on Kanban Cards</i>	77
<i>Figure 17: Show Dependencies Between Different Teams</i>	77
2.4 Committing the Perfect Crime—Unplanned Work	
<i>Figure 18. A Study in Interruptions</i>	85
<i>Figure 19. A Study in Pink Dots</i>	86
<i>Figure 20. Expose Unplanned Work</i>	87
<i>Figure 21. Monthly Delta Trend for Unplanned Work</i>	88
2.5 Prioritize, Prioritize, Prioritize	
<i>Figure 22. An Experiment in Tagging and Prioritizing</i>	96
<i>Figure 23. A3 Example</i>	97
<i>Figure 24. Exposing Conflicting Priorities</i>	99
<i>Figure 25. Inputs That Contribute to Cost of Delay</i>	100
<i>Figure 26. Cost of Delay</i>	101
<i>Figure 27. Line of Commitment</i>	102
2.6 Preventing Negligence	
<i>Figure 28. The Validate Pit</i>	106
<i>Figure 29. Expose Neglected Work</i>	109
<i>Table 2. Conflicting Priorities Table</i>	112
2.7 Useful Board Design Examples	
<i>Figure 30. Multilevel Board Design</i>	115
<i>Figure 31. Done vs. Done Done</i>	116
<i>Figure 32. Plan-Do-Check-Act Board Design</i>	117
<i>Figure 33. Home Project Board</i>	118
<i>Figure 34. Manage Your Move Board</i>	119
<i>Figure 35. Repetitive Tasks</i>	120
<i>Figure 36. Purchase Order Board Design</i>	121
<i>Figure 37. Student Board</i>	123

PART 3 METRICS, FEEDBACK, AND CIRCUMSTANCES

<i>Figure 38. Teams Within Teams Board</i>	127
3.1 Your Metrics or Your Money	
<i>Figure 39. Flow Time Metrics</i>	133
<i>Figure 40. Lead Time and Cycle Time</i>	134
<i>Figure 41. The WIP Report</i>	135
<i>Figure 42. Queuing Theory</i>	137
<i>Figure 43. Aging Report</i>	138
<i>Figure 44. Flow Efficiency</i>	139
<i>Figure 45. Optimal Batch Size</i>	140

3.2 The Time Thief O'Gram	
<i>Figure 46. The Original Time Thief O'Gram</i>	143
<i>Figure 47. Congregated Time Thief O'Gram</i>	144
<i>Figure 48. Balanced Scorecard</i>	145
3.3 Operations Review	
<i>Figure 49. Cumulative Flow Diagram for Ops Review</i>	149
3.4 The Art of the Meeting	
<i>Figure 50. Lean Coffee Setup</i>	155
3.5 Beastly Practices	
<i>Figure 51. Individually Named Swimlanes</i>	165
<i>Figure 52: T-shaped Skills</i>	166
<i>Figure 53. Specialization</i>	166
Conclusion: Calibration	
<i>Figure 54. The J Curve</i>	173
Afterword to the Second Edition	
<i>Table 3. The Skills Matrix</i>	183

Preface to the Second Edition

My main aim with *Making Work Visible* was to help teams (IT Operations teams in particular) navigate problems related to conflicting priorities, dependencies, neglected work, and the ominous too-much WIP.

A lot has happened since the first edition of *Making Work Visible* was published in 2017. I've learned even more about workflow and the impacts of poorly managed value streams on people and businesses.

I'm going to address some of these additional learnings (and make some corrections) that I've come across along the way that would be helpful for you to think about and know as you dive into the rest of the book.

Let's begin with a major area of concern that needs more attention—the need to acknowledge and address the lacking partnership between technology and business leaders. It can be fairly easy to optimize workflow within the control of your internal teams, but meaningful improvement at the business level without mutual trust and partnerships is a tough journey. For significant improvement, you need a coalition of the willing—a broader community of real buy-in and support from business leaders.

The Business Relationship

One area I briefly covered in the first edition was the topic of technical and business team alignment. This topic deserves further attention. I

didn't describe the essential need to partner with business people and the need to fully understand business perspectives. If companies hope to get better at what they do, then technology must become a strategic partner to the business. Torn relationships need diligent attention to advance beyond shared accountability and toward shared goals, mutual respect, and true collaboration.

The better you understand the context in which the business operates, the better you can function as a major player to business leaders in identifying opportunities for technology to gain a seat at the table as a strategic partner.

It is useful to take stock of your own understanding of your company. Ask yourself: Do you truly know your business? Then consider the following questions:

- What problems do we solve for our customers?
- How do we specifically solve those problems?
- How do we do it differently from our competition?
- What is our proof? How do we validate our credibility?
- How does our company's financial performance compare with the rest of our industry?
- Why do our customers choose our product or service over other options?

These questions speak to the core of your business. Your business counterparts have a responsibility to drive business outcomes, so these are the things that your business stakeholders obsess about daily. Technologists who lack an understanding of the business context risk investing in initiatives that ultimately won't drive value. Studying your business puts you in a better position to gain trust from business leaders and to coauthor and prioritize business objectives.

If you try and try but don't make progress, pause to consider the possibility of irrational perseverance (the failure to abandon a situation when available evidence suggests otherwise). If, at some point, it dawns on you that a business partnership is implausible—that it's too hard to continue going at it alone—then perhaps it's time to seek partnerships elsewhere. If support is nowhere to be found in your company, find like-minded communities of practice outside your company—like

at the DevOps Enterprise Summit! Come join us. We're the ones having all the fun anyway. :-)

Doing it All

Product teams in the midst of change at one of the biggest US insurance companies came up with an experiment they called “Doing it All.” The sarcastic title was intentional, as it magnificently described the nature of a theme that I observe across multiple industries. From finance to insurance to transportation, carving out capacity for you and your teams to implement a new way of working requires herculean efforts.

Change takes time. It's not enough to just give teams approval to do something. They also need time allocated to do so during regular business hours. Some people at the insurance company I mentioned above were working sixteen-hour days to meet expectations. Leadership may agree to change and approve the budget for a transformation, but if teams are not given the capacity to learn new tools and processes on top of their existing load, then “doing it all” burdens the teams with too much cognitive overload, creating resistance and resignation due to the J-curve effect discussed in the conclusion of the first edition of this book.

Those struggling with conflicting priorities and caught up in “doing it all” can improve their situation by helping others understand the impacts to their business bottom line using metrics. Measuring the value you're delivering for the business is a useful way to address common objections to develop a healthier, more resilient partnership.

When people complain that things take too long, technology leaders should be positioned to demonstrate just how long things actually take to do in a way that others can see, helping to show that “doing it all” is not a good strategy. When the VP of Sales (or anyone else) claims that things take too long, it makes sense to measure and show how long things actually take. This measure of speed (flow time) is helpful to start a data-driven conversation on objections and offer a thoughtful response. This and other objections that you might encounter are listed in Table 1, along with suggested responses and useful metrics to use when building your argument.

Table 1: Prepare for Objections with Key Metrics

OBJECTION	RESPONSE	KEY METRIC
"Things take too long."	"Here's how long features actually take to do."	Flow time: measures speed. Clock starts when work begins and ends when the customer can consume the value.
"Not enough features get delivered."	"The capacity of the teams is often less than the demand. Teams are constrained by conflicting priorities."	Flow velocity: measures throughput—the number of items completed over a period of time; this is useful for gauging capacity.
"We should just outsource this to vendors."	"Outsourcing this work will likely insert more dependencies into the workflow, which impacts efficiency due to back-and-forth communication delays."	Flow efficiency: the ratio of wait time vs. active time. Prompts the reasonable question, "Do we need more wait states in our toolset to accurately reflect all the wait time in the value stream?"
"We need features to be delivered, but you just want to refactor architecture."	"We need to change the oil in the car every now and then. Otherwise, we won't have a car to drive."	Flow distribution: the distribution of different work item types, e.g., 30% features, 60% defects, 10% debt.
"We are at risk of not completing all the initiatives on this year's roadmap."	"Attempting to do too many things at once results in lower quality and slower delivery. It's why Steve Jobs killed many great ideas at Apple, so they could do fewer things really well."	Flow load: the amount of work started but not yet finished. Lots of partially completed work-in-progress (WIP) is expensive.

It may seem as though conversations like these are adversarial. However, they are essential for partnership building discussions. Every good objection has a worthwhile objective to consider. As objections come up from business leaders, be prepared to address these and see other people's perspectives. Two people may bring their opinions to the table. The outcome is that they both walk away with a third option that is better than either of the original two. Consider different perspectives as feedback to move toward a shared path forward.

More on Best Practices

To the disgruntlement of some people, I have been skeptical of best practices since I was first exposed to the idea. But now more than ever, I fear that teams strive to implement and follow best practices in situations where novel ideas are needed instead. The assumption that published works are the correct approach to take is an interesting topic. I acknowledge the irony of this statement given I have suggested practices and guidance in this book.

It's just that people seem to read things and assume or accept it as a best practice. But best practice is a misnomer. I suggest a reframe. Remove (or judiciously reduce the use) of the term "best practice" from conversations about what to do.

This book offers good practices. Don't think of them as best practices that are never to be improved upon; think of them as optional tools available in your toolkit. Recognize that best practices are frequently usurped by better emerging practices. Because of rapid technical and process change and the need to evolve, your organization's best practice may not be a best practice for very long. One of the best skills you can have going forward is learning how to keep pace with an ever moving edge.

We need diversity of thought and different ways of working. If we get caught up in the idea that there is a best practice—a perfect way of doing things—then we risk branding fresh perspectives or improvements as inherently flawed ideas. This is problematic in a period of time when things are changing so rapidly and we don't always know

the effects or causes that occur. Creativity occurs at the edges. Looking at problems from the fringes of recognizable boundaries with new and novel ideas can be helpful for organizations looking to innovate and to experiment.

I am blessed to have such a person on my team. They refreshingly introduce me to new ideas and concepts, which are met with much enthusiasm. It's important for varied thoughts and different perspectives to be heard as acceptable ideas for consideration. It's a good thing when people have energy and passion for new and novel ideas because 100% investment in one area is risky business. Consider the necessity for those working on the edge with different opinions and ideas to occur.

Nip the assumption in the bud that if it's not a best practice, it's bad. Practices are always changing. We don't live in a static world. Call them "today's practice" or "this week's practice" or our "2022 practice," anything but best practice.

Corrections

In the first edition of *Making Work Visible*, I got some details wrong. Take Unplanned Work. I need to acknowledge that Unplanned Work isn't really a work item type in itself because any type of work can be unplanned. It's still a time thief because of the resulting interruptions and context switching, but I no longer consider it a specific work item type. There can be unplanned features, unplanned tech debt, unplanned security vulnerabilities, and unplanned defects. This is similar to experimental types of work—there can be experiments around features, defects, risks, and debts too. If you want to notate work as unplanned or as an experiment, then tag it or flag it as such in your toolset to make these visible in your metrics.

I've also learned more about vision. There is so much more to vision than colors, shapes, and textures. The eyes are the only two parts of the brain that reside outside of the cranial vault. They are part of the nervous system and are how we perceive where we are in space and time. Our vision has everything to do with our psychological safety. Sensing

that we are okay in a virtual meeting where people don't have their cameras on makes it almost impossible to tell how others are responding to our words. Are they nodding their head in agreement? Or are they rolling their eyeballs in disgust? I think it would behoove us all if people turned their camera on at least while they are speaking.

The line of commitment (Figure 26) is something that I've also reconsidered due to the planning fallacy—where plans and forecasts are unrealistically close to best case scenarios. People generally make decisions based on unrealistic optimism. They overestimate benefits and underestimate costs. As a result, they pursue initiatives that are unlikely to come in on budget or on time or deliver the expected return or even be completed. I suspect the probability of work passing the line of commitment and still being canceled (or not even used if indeed delivered) due to changing priorities is higher than we think. Small batch size and fast feedback can delay decisions to continue building out a request way beyond the traditional line of commitment. When/if ideas are positioned as experiments, the value of a line of commitment seems moot—just a place to start the clock? We need to start the clock somewhere I suppose, but I retract my inept comment about no second guessing once work starts (see page 100).

Please keep the above in mind as you read the rest of the book. Thank you for your time (that time you didn't have to read this). Just joking. :)

—Dominica DeGrandis
Woodinville, Washington
December 2021

Day, n. A period of twenty-four hours, mostly misspent.

—Ambrose Bierce

FOREWORD

So about that internet meme, the one assuring our frazzled selves that everyone has the same twenty-four hours in their day as <insert entrepreneurial rock star here>.

I'd like to nip that bit of condescension in the bud and offer an emphatic, not quite. While many of our business role models are in fact driven by a seemingly superhuman work ethic supported by 100+ hour work weeks, they nevertheless have an advantage over us mere mortals. While the number of minutes available to us each day might be the same, control over what we do with those hours differs significantly. When Elon Musk is faced with too much work-in-progress (WIP), he has the authority to delegate, deprioritize, or simply say no. When variation rears its head and a well thought-out strategic plan no longer aligns with the organization's needs, Sheryl Sandberg has the ability to switch gears. And when Jeff Bezos is confronted with conflicting priorities, it is likewise doubtful that he needs to seek direction via a convoluted bureaucracy to gain clarity over which course to follow.

When these things happen to us (and let's face it, they often do), we're faced with a very different set of repercussions than those of our billionaire counterparts.

So what about us? In the absence of unbridled agency and an extensive support staff, how do we do all that needs to get done, when it needs to get done, without sacrificing quality or our sanity in the process? In a culture that exalts productivity and perpetuates the mythology of multitasking, how do we maximize our time and our workflow to the point that our effort and our energy yields the greatest impact? Most importantly, how do we do all of that and still have time for living?

Time saved. Time spent. Time wasted. We frame conversations about time much in the way we do money. Ostensibly “free” but nevertheless invaluable, time is arguably one of the most precious resources we have, yet one we never seem to have enough of as individuals, as teams, or as organizations.

Anyone who has ever been faced with a deadline can certainly relate to Parkinson’s Law: work expands to fill the time available for its completion. Let’s be honest—when was the last time you completed quality work days or even hours ahead of deadline?

You’re not alone.

It seems we’re constantly doing. But doing what exactly? Why are our weeks filled with days where we return home exhausted, only to lament how we’ve barely made a dent in our to-dos? Like the elusive sock that mysteriously goes missing in the laundry, where do those lost hours go? Who—or rather what—is responsible for stealing our time, our focus, our energy?

The attempt to harness or “keep time” is in no way a modern or even premodern convention. Prehistoric humans tracked the phases of the moon. The Sumerians created the sexagesimal numeral system still in use today, employing sixty to divide the hour into minutes and then minutes into seconds. The Egyptians used obelisks to calculate the length of shadows cast by the sun. The shortcomings of solar-based measures became apparent the moment clouds appeared or the night sky arrived, and so with clepsydras—or water clocks—the Persians and Greeks offered an alternative, monitoring water flow instead to measure the passage of time.

With these ancient time-tracking tools came the earliest forms of scheduling—when to plant crops and bring in the harvest, when to conduct commerce, and when to perform daily rituals such as eating or sleeping.

Fast forward to today. Despite all our modern “conveniences,” effective time management has for many become an uphill battle, an all-consuming if not quixotic goal. While the Information Economy ushered in 24/7 connectivity, it likewise begot round-the-clock expectations, and so, paradoxically, technology like mobile phones and email and video conferencing—tools that would ostensibly make life

easier—often enslave us. We allow the chaos of modern work coupled with an often paralyzing number of options at our disposal to overload us, to distract us, to stealthily steal our time and focus, and ultimately impede our effectiveness.

We tend to fetishize the complex. But just as the earliest solutions for tracking time were both easily implemented and yielded extraordinary results, so too are the ideas detailed in *Making Work Visible: Exposing Time Theft to Optimize Work & Flow*. Much in the way sky, sun, sticks, and sand provided ancient man actionable, visual feedback, so too do the suggestions Dominica outlines in the pages that follow.

It should come as no surprise that we can better manage what we can see. When we can't see our work, our options are obscured. We're blind to our own capacity, and we certainly can't communicate that capacity to others. The resultant mental overload creates stress. Stress compounds the work we already have, essentially contributing to WIP, compromising the ability to focus, prioritize, make decisions, and complete work with quality, let alone complete work at all.

The visualization and WIP-limiting strategies Dominica offers demystify our cognitive load; normalize expectations among team members; promote focus; situate work in its context, surfacing problems (and allowing for solutions to be made) in real time; and provide a clear path to completion with quality. Elegantly explained and deeply insightful, the utility of the suggestions contained within cannot be overstated.

To be sure, it is no small irony that I'm writing about "time theft" while spending a week aboard a sailboat exploring an archipelago in the Salish Sea beholden only to "island time." It's the first holiday where I've intentionally left my watch at home, instead choosing to be fully present to the wildlife and seascapes around me: bald eagles and peregrine falcons soar high above craggy bluffs where the pristine coastline meets old-growth forests. Otters gracefully propel themselves through the glassy surf, disrupting kelp and eelgrass in search of their next meal. Along the rockier parts of the shore, scores of sea lions loll in the sun as hauled-out harbor seals nurse their spotted pups on a secluded beach nearby. A congregation of boats idling in the distance serves as a familiar sign, and it's not long before I too glimpse a family of orcas perform for their Nikon-wielding audience (affectionately known by locals as the

“pod-parazzi”) on crystalline jade waters that seemingly end at a cerulean blue sky.

If there was ever a place where I’ve given less thought to watching the clock, it is in this Pacific Northwest jewelbox known as the San Juan Islands.

This is precisely why Dominica’s book is so important. Our culture of overwork, our obsession with productivity versus effectiveness, our default mode of existing rather than living—these things aren’t simply unnatural and unhealthy, they’re unsustainable—for the individual, for the team, for the organization’s bottom line.

The thoughtful observations and easily implementable suggestions Dominica offers are the first step in helping create habits that lead to a virtuous cycle of healthy, sustainable, and improvable work. Work in which we experience more clarity, less stress, increased focus, improved decision-making, manageable workloads, and, by extension, a more fulfilling work day, which in turn affords us the slack needed to fully live our lives rather than simply chase productivity.

So while technically we have the same number of hours in a day as <insert entrepreneurial rock star here>, it’s creating thoughtful work systems that make us cognizant of what we use those hours for during the workday. And it is what those hours afford us the opportunity to do when we leave the office that makes for a fully integrated life.

Indeed, time is sacred. Treat it as such. Visualize your work. Limit the amount of work you take on. Pay attention to its flow. Build thoughtful work systems to reflect what really matters.

To breathe. To think. To learn. To grow. To play. To love. To live.

For it is in working well that we can live well. I am confident the wisdom Dominica offers in the pages that follow is the first step to achieving such existence so that you too can begin to experience less stolen time and instead plan for more island time.

—Tonianne DeMaria
Orcas Island, Washington
2017

Do not squander time for that is the stuff life is made of.

—Benjamin Franklin

INTRODUCTION: WORK AND FLOW

As a build engineer, my first job out of college was to make builds visible. This meant tracking which version of what file went on which computer and in what environment. Three months into that job, I was working on a build—getting all the code from the source code repository, compiling it into executables, and then installing the resulting new functionality into a place where others (analysts, developers, testers, and other interested people) could see it. The build wasn't compiling though, and I sat there troubleshooting the broken build in the office, alone, at 2:00 a.m. Tired, I was making mistakes, so I went home. I seriously questioned my career choice. Technology work apparently meant working many late nights. After a nap, I returned to the office to track down code dependencies between various developers and eventually got the build working.



I'm not sure exactly how many hours I've spent tracking down dependencies during all my years of merging, building, and releasing software, but I'm convinced that it's been way too many. If I had a dollar for every minute spent troubleshooting builds and broken environments, I would have a sweet little nest egg. Delayed work, whether it is measured in hours, days, weeks, or even months, carries with it a cost. Losing time due to avoidable problems is expensive and dispiriting. Life is short. Wasted time can never be regained.

In the sci-fi film *In Time*, time is literally money. People earn minutes, hours, and days to buy food, housing, transportation, and everything else imaginable. Street thugs kill people by stealing all their minutes.

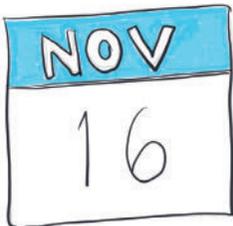
Wasted time is the kiss of death. In one memorable scene, Will Salas, played by Justin Timberlake, saves the life of the wealthy Henry Hamilton, played by Matt Bomer. When Will and Henry get to a safe place, Henry tells Will that he is 105 years old and tired of living. He asks twenty-eight-year-old Will what he would do with 100 years. Will quips back, “I sure as hell wouldn’t waste it.” Later, as Will sleeps, Henry gives Will his 100 years and leaves him a note, “Don’t waste my time,” before he runs off to timeout by allowing his own clock to run down while sitting on the ledge of a tall bridge.¹



A scrawled note from a dystopian sci-fi film embodies our reality. Time is life—use time wisely.

We workers are drowning in nonstop requests for our time. From developers to IT operations people, it’s overwhelming to keep up with the ever increasing demand. In this regard, things haven’t changed much since my first job out of college, where as a software configuration management lead at Boeing, I did builds and deployments on IBM mainframes at Hickam Air Force Base in Hawaii.

A line of people formed outside my cube wanting to know the status of a build. “Did everything compile okay?” “When will the build be deployed to the quality assurance environment?” “Can I get one last change in?” I wanted to say, “Pick a number. I’m working as fast as I can. Every one of your interruptions delays the build by another ten minutes.” The fact that developers and testers had come to me to ask for status updates was a symptom of a much larger problem that I didn’t recognize at the time.



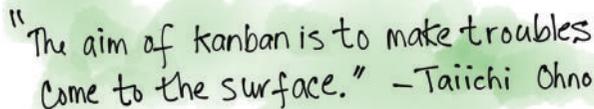
My calendar was booked with meetings all day long. Rarely did I get a chance to work uninterrupted until the evening or the weekend. Four months into the new job, I pulled an all-nighter in the office, working as fast as possible to catch up on the mountain of work. When the program manager arrived at 6:30 the next morning, he thought I had just arrived early. He was not pleased to hear that I was headed home to take a nap. Sleep deprivation was another red flag that I didn’t give enough thought to at the time.

Later on, after years spent working in technology, I recognized that relentless heroism—staying late night after night, wearing two hats, and constantly playing catch-up—is unsustainable. Quality doesn't happen on four hours of sleep.

We overload ourselves and we overload our teams. This is the everyday reality within the information technology sector. And, because we get interrupted all the time, we stop work on one task and start work on a different task, from one project to the next, never focusing on one thing long enough to do it justice. This context switching kills our ability to settle into work and concentrate sufficiently. As a result, we are unhappy with the quality of our work despite our desire for it to be good.

The problem is that we are working with dysfunctional processes. Companies haven't adapted to keep up with demand in a healthy, sustainable way. Instead, we see the continued use of antiquated approaches meant to keep workers busy all the time. These processes are not working. This is the elephant in the office. If workers were able to get everything done right and on time, there wouldn't be an issue. But that's about as common as a black swan. The amount of requests (the demand) and the amount of time people have to handle the requests (their capacity) is almost always unbalanced. This is why we need a pull system—in which people can focus on one thing long enough to finish it before starting something new—like kanban.

Kanban is a visual pull system based on constraints that allow workers to pull work when they have availability instead of work being



"The aim of kanban is to make troubles come to the surface." —Taiichi Ohno

pushed onto them regardless of their current workload. Since demand and capacity are frequently unbalanced, and it's almost impossible to get everything done on time, systems like kanban are for helping people balance all their work demand.

We'll get into kanban and where it fits into the process of making work visible a bit later, but for now, know that kanban is an approach to make work and problems visible and improve workflow efficiency. Kanban helps you get work done efficiently without burning the midnight oil night after night.

In the 2000s, I worked at an image licensing company in Seattle owned by Bill Gates called Corbis. I managed the Build and Configuration Management team. We had a decent reputation among the Engineering department until 2005, when our two preproduction, seven-server environments quadrupled into eight preprod, twenty-five-server environments. We had seventeen databases. Each configured manually within the tightly coupled, highly dependent architecture. On top of that, the business asked us to develop new major systems at the same time, and they wanted the ability to deploy either one before the other. The dependencies between the existing system and the two new systems ballooned. My job grew from building out and managing twenty-five servers to building out and managing two hundred servers.

To deal with the changes, we created and maintained additional long-living branches in source control, which is the place where developers check in their code for safekeeping. It was a terrible solution, but it helped the teams avoid clobbering each other's changes. Think of long-living branches as a place where code is stored in isolation, where it's impossible to see the impact it might have on the code already released to production. It's kind of like adopting an older cat and praying that he and your current, much older cat will embrace each other with open paws. With more than two hundred servers to configure and maintain, configuration management was elevated. It took, at best, two weeks to restore production data to preproduction environments. We scheduled "M Is for Merge" days every six weeks, which consumed many developers' time.

Our reputation plunged. Developers complained that builds were taking too long. This, of course, offended me, and I set off to prove them wrong by collecting build-and-deploy time metrics.

I pointed out that the big ball-of-mud architectural design disaster on our hands made deployment and maintenance of environments problematic. I pointed out that the manual smoke tests (tests to see if website functionality still works) delayed the time that developers and testers

could see the latest changes and that lack of automated testing hurt our ability to quickly spot problems. Manual smoke tests were the norm. Both of these problems were dismissed fairly quickly as not real issues. The fact remained that developers and testers were unhappy. Business people were unhappy. And the boss was unhappy. It's no fun being on the team that "doesn't deliver." The barriers between teams were stronger than the connections. This is the problem with a bad system.

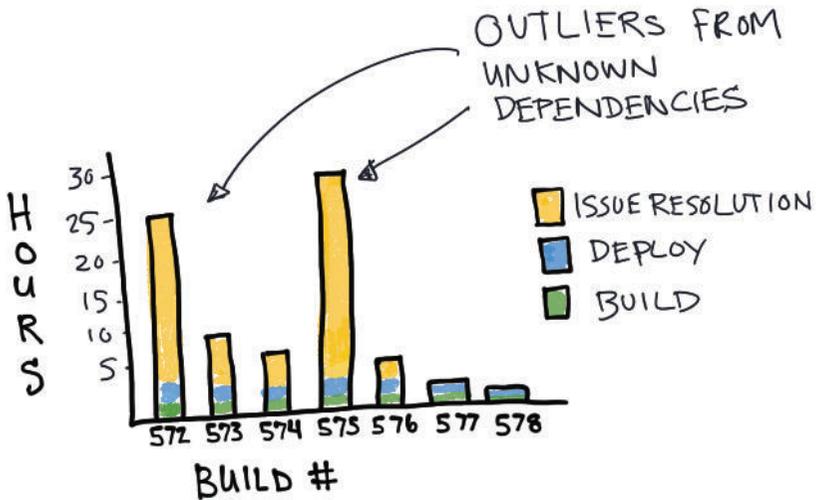


Figure 1: Builds Don't Take That Long

My own experience with a bad system coincided with the CFO deciding to replace the enterprise resource planning (ERP) system with another ERP product called SAP. An ERP system is a management information system that integrates planning, purchasing, inventory, sales, marketing, finance, HR, etc. SAP is its own ERP system, created by SAP AG, the fourth-largest software company in the world.

My boss asked me, "Hey, do you want to manage the SAP Basis team as part of managing the build and release team?" Like an idiot, I said yes. I don't know how I could have possibly set myself up for more failure. I had zero experience with SAP, and adding SAP to my list of responsibilities spread me thin—to the point where I managed to be terrible at many different jobs. Multitasking is a good way to screw up progress, as I'm sure many of you reading this book know from experience.

At the time, I didn't know that all these things were red flags of a bad system. All I saw was that my performance was less than exemplary and that I was an unhappy employee who had started to consider other options.

I updated my resume.

In 2006, we spent a good deal of time analyzing and comparing different tools to manage our source code. Our team chose Team Foundation Server (TFS). We were a Microsoft shop, after all, and I ended up installing, configuring, and maintaining TFS—while also learning SAP, interviewing new candidates weekly, and helping to implement a new sustainment process. This process made it possible for us to deliver improvements every two weeks instead of every six months.

A user interface (UI) developer named Dwayne Johnson recognized the value in delivering small changes frequently and began socializing the idea of making small improvements on a consistent schedule. Dwayne started the process by fixing UI bugs on a regular bi-monthly cadence. At the time, it was just one more thing to support, but it was a very important one. These incremental and iterative improvements done on a regular cadence were our Agile alternative to traditional Waterfall development. These Agile methods wandered into our process, getting us thinking about a better approach to our work.

In April of '06, a Scottish Fellow from Microsoft appeared at Corbis. David J. Anderson visited us monthly to teach us how to apply the Theory of Constraints (TOC) to our work in exchange for permission to write a story about the Corbis Agile transformation.

TOC is a way to identify the most important limiting factor (the constraint) that stands in the way of achieving a goal and then systematically improving that constraint until it is no longer the limiting factor. There was much excitement while reading his book *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results* as we thought we would do Feature Driven Development, a type of Agile development focused on cross-functional, collaborative, and time-boxed activities to build features.



As Darren Davis writes in his blog post “The Secret History of Kanban,” David’s methods “. . . eliminated explicit estimation from the process, and relied on data to provide a probabilistic means of determining when software was likely to be done.”² David got us going on operations reviews and explained how important it is to measure progress (or lack thereof). Learning what to measure changed my world. Ranting didn’t work but measuring cycle time (the time it takes to do work) and presenting that data to leadership, did. I was able to influence leadership then and got buy-in to hire additional team members.

Sometimes the obvious gets lost in the crunch of the corporate world. We intuitively knew we had too many projects in flight, but it was hard to see until we measured the actual time that it took to get work done, at which point it became obvious that the work spent more time in wait states than in work states. We spent time waiting for approval. Waiting for others to finish their part so we could start (or finish) our part. Waiting for uninterrupted time to focus on finishing the work. Waiting for the right time of day/week/month. And while we waited, we started something new, because, you know, with resource utilization as a goal, you have to stay busy all the time.

As Kate Murphy writes in her article “No Time to Think,” “One of the biggest complaints in modern society is being overscheduled, over-committed and overextended. Ask people at a social gathering how they are and the stock answer is ‘super busy,’ ‘crazy busy’ or ‘insanely busy.’ Nobody is just ‘fine’ anymore.”³

I see evidence of this every day. When there is a still moment for reflective thought—say, while waiting for a meeting to begin—out come people’s phones. Busyness can be an addiction for terminally wired ambitious people. But busyness does not equate to growth or improvement or value. Busyness often means just doing so many things at once that they all turn out crappy. Sometimes walking in the park and allowing ourselves time to think is the best way to seize the day. But horrors if an engineer sits idle for fifteen minutes simply thinking.

At Corbis, looking at the reasons why we worked on too many things at once was a revealing exercise. The CFO wanted to implement a new financial system. The SVP of Global Marketing wanted to blah, blah, blah. The VP of Media Services also wanted blah, blah, blah. The head of

Sales wanted blah, blah, blah, blah. And they all wanted everything now. The resulting business priorities clashed all the way down the hierarchy, and that was just the business side of the house.

On the engineering side, not only did we need to implement all the business requests, we also had our own internal improvements to make and maintenance work to do. Furthermore, we still had to be available to drop everything when production issues occurred—like it or not, production comes first.

The clashing priorities became apparent while looking at the many long-standing branched code lines, but other than that, there was no clear visual of the impact of working on too many things at once. It's hard to manage invisible work. With invisible work, we don't notice the explicit reminders that our mental budget is already full. There is no time to simply think.

After eight years at Corbis, I was one of forty-two people let go during the September 2008 round of layoffs. At this point, I decided to try something different. I got a job with AT&T Mobile on their program management team. But the regression from using the Lean kanban approach I helped create at Corbis to using a Waterfall approach (a traditional software development method where work waits until all the parts of the previous stage are complete), with estimations based off of time reports, was too much of a throwback for me. In July 2010, I fired myself.

In January 2011, David J. Anderson offered me the opportunity to research, develop, and teach a new course for David J. Anderson & Associates called “Kanban for IT Operations.” At the time, Europe led the United States in kanban implementations, so my research in February of 2011 began in England, Sweden, and Germany. In March, we ran the first beta workshop in Boston, where I attended and spoke at DevOpsDays Boston 2011 at the Microsoft New England Research and Development Center.

Originally, I set off to write a reference for students to use during workshops while designing their kanban boards. Later, this piece grew into a time-saving reference for me as well. It became a place to capture not only everything I learned about applying Lean, kanban, and flow practices to my own work but also included selected equations, theories, and stats from thought leaders.

For example, how to define Lean? For that, I prefer Niklas Modig and Pär Åhlström's definition. In their fantastic book *This Is Lean: Resolving the Efficiency Paradox*, they define Lean as, “a strategy of flow efficiency with key principles of just-in-time and visual management.”⁴

So, what do we know? We know the demand for delivering business value to production so that we can be competitive is high. We know that many organizations are running deployment strategies that are slow and cumbersome. We also know that we are wired to do our best when we can clearly see what we are doing right as well as what we are doing wrong. This might seem obvious, but it's consistently ignored.

The technology world shows no signs of slowing down. The pace at which we need to deliver new capabilities to win new customers and prevent existing customers from walking away (churn) seems like warp speed. Many companies today are in survival mode, they just can't see it. This means there is no better time than right now to elevate how we work. So, how do we level up our game?

The answer is straightforward and accessible. It doesn't cost you tons of money, and it doesn't take geniuses or specialists. All it takes is a shift from haphazardly saying yes to everything to deliberately saying yes to only the most important thing at that time. And to do it visually.

The solution is to design and use a workflow system that does the following five things:

1. makes work visible
2. limits work-in-progress (WIP)
3. measures and manages the flow of work
4. prioritizes effectively (this one may be a challenge, but stay with me—I'll show you how)
5. makes adjustments based on learnings from feedback and metrics

In this book, we will cover:

- How to spot the five thieves that steal your time.
- How to expose the time thieves in order to make work visible and optimize workflow.

- How to know how you are really doing using metrics and feedback.
- What practices can get you into trouble.
- How to influence leadership decisions.

The examples described throughout the pages of this book are all based on my own real-life experiences and on the experiences of others who have stood as witnesses to time-theft scenarios. Some prefer to avoid publicizing the crimes committed within their companies, so for them, the names have been changed to protect both the innocent and the guilty. We will also be looking at systemic organizational issues that must be addressed in order for you to be successful. As Edwards Deming said, “A bad system will beat a good person every time.”⁵

This book is simultaneously an explanation, a how-to guide, and a business justification for using Lean, kanban, and flow methods to increase the speed and effectiveness of work.

Everything in this book may not apply to your specific situation. It has an IT bent to it, with several non-IT examples thrown in for good measure. Take what does apply to you and use the rest to gain insight into what people in other parts of your organization, or your competitors, might be dealing with.

Each section in Part 2 includes exercises from my workshops, where we step through a series of activities designed to make work visible, improve workflow efficiency, and surface problems. They build upon each other, so it is best to read the sections in sequential order.

Explaining the concepts in this book to others should be a straightforward process. Getting buy-in to implement the suggested approaches may not be. Change is hard for humans. So, before we dive into workflow design, let’s investigate exactly what prevents you from getting your



work done quickly in the first place. Once we scrutinize the crimes committed against your existing workload, we can proceed with the insight and awareness necessary to do something about it. Let's get started.

PART 1

Stealing, of course, is a crime, and a very impolite thing to do.

—Lemony Snicket, *The Wide Window*

THE FIVE THIEVES OF TIME

If your wallet was stolen, you'd notice. If your security badge to your office was filched, you'd know it when you arrived. And if you opened the fridge to find your lunch missing, you'd make sure your office mates heard about it. So why don't people notice when they are robbed of something more valuable than their wallet, badge, or lunch—their non-renewable time?

We grumble that there just aren't enough hours in the day and that someone else sure seems to have a lot of free time. But we regular mortals only have twenty-four hours in a day. The problem is that we don't protect our hours from being stolen. We allow thieves to steal time from us day after day after day.

Who are these thieves of time?

The five thieves of time that prevent you from getting work done include:

1. Too Much Work-in-Progress (WIP)—work that has started, but is not yet finished, sometimes referred to as partially completed work.
2. Unknown Dependencies—something you weren't aware of that needs to happen before you can finish.
3. Unplanned Work—interruptions that prevent you from finishing something or from stopping at a better breaking point.
4. Conflicting Priorities—projects and tasks that compete with each other; this is exacerbated when you are uncertain about what the most important thing is to do.
5. Neglected Work—partially completed work that sits idle on the bench.

These five thieves hide right under your nose, comfortably cozy between you and your work. But they leave clues at every crime scene. If we're going to get stuff done, we must trap these thieves to expose the crimes they commit. Once the thieves are caught, we can begin to do something about their insidious wreckage. Instead of being at their mercy, we can turn that dark corner, take back control, and make the kind of improvements that matter.

1.1

Beware the barrenness of a busy life.

—Socrates



TOO MUCH WORK-IN-PROGRESS

On the roof of a building, Saturday, 9:00 a.m.

A man undertakes an item on his honey-do list (his to-do list, heavily influenced by his spouse), which includes dismantling the roof of an outbuilding. Over the years, this man's list of things to do has included repairing everything from appliances to septic systems. He has buried power lines, felled ninety-foot-tall cedar trees, and built cabins and garages step-by-step from excavating foundations to installing flooring, heating, plumbing, electricity, and roofing.

Recently, he seismically retrofitted an unreinforced, hollow-tile foundation. I am this man's assistant, which includes (but is by no means limited to) acting as tape measure holder, safety inspector, and demolition and cleanup crew.

One day, while helping him tear down the rafters of an old, crumbling 24x36 foot outbuilding (I on the ground, he on the roof), I casually suggested that we build a 16x24 foot greenhouse on the back forty. From the top of the twenty-five-foot-tall rotting roof, my beloved husband looked at me incredulously and said, "Hon, can't you see I'm busy up here?!"

The tech sector does not have a monopoly on too much work to do. Talented people everywhere receive long to-do lists. The problem for a spouse who can build or fix anything is that the other spouse provides them with a long list of



things to do. And it's hard for them to say no (unless they are atop a twenty-five-foot tall rotting roof).

We humans have a hard time saying no for a variety of reasons. Reason number one is because we like the person who asked us. The same



holds true at the office. Because network engineer Sean gives me a heads up on work coming down the pipe that impacts me. I say he's nice and am willing to help him out when he needs something. But Carlos! Carlos knew about this port change two weeks ago and is only just now telling me at 5 p.m. on Friday?! My mental narrative says, "I don't really want to help you."

There are six main reasons people give when I ask them, "Why do you take on more work than you have the capacity to do?" We like the person who asked—as I pointed out in my earlier example. We are team players—"I don't want to be the person who lets the team down."

1. We fear humiliation—"I don't want to be criticized or fired." Yes is easier to say than no—especially to the boss. Also, refusing a manager's request can be risky in some cultures.
2. We like new and shiny—It's much more fun than doing the grunt work it takes to finish something complicated and unglamorous.
3. We don't realize how big the request is until we start working on it—"Oh, no problem. I can get that done in just a couple of hours," but the task takes much longer.
4. We like to please people—"I say yes to most requests in general because I want to be liked, admired, respected."

Vanessa Bohns, a social psychologist and professor of management sciences at the University of Waterloo in Ontario, says, "It comes down to this fundamental motivation we have to stay connected to other people. We don't want to reject people. We don't want people to think poorly of us... so we are really managing the impressions other people have of us."¹ On the flip side, we rarely realize the power we have over other people when we ask them to do something, especially if others worry about explicit or perceived positions of power.

In textbook terminology, too much work-in-progress (WIP) is when the demand on the team exceeds the capacity of the team—which is a rather boring way to say that our teams are drowning in work, often because their schedule is completely full. Every minute of the day is fully scheduled (or fully allocated to 100% resource utilization). The most talented have the longest lists. This equates to people doing their full-time job on top of everything else that is expected of them, such as troubleshooting environment issues (problems with the configuration of servers that prevent website functionality and other things from working right), hiring new team members, and completing merit reviews to name just a few. Similar to how our digestion system lets us know when we've stuffed too much food down it, Thief Too Much WIP attacks us if we cram too many meetings into our day, leaving us unable to begin the day's to-do list until 6:00 p.m.

Why Too Much WIP Matters

Too much WIP matters for a number of reasons. It can result in many issues, including delayed delivery of value,



more no, less wip

increased costs, decreased quality, conflicting priorities, and irritable staff, to name a few. When we start a new task before finishing an older task, our WIP goes up and things take longer to do. So does the potential loss from things taking too long to complete and being unable to realize the value of it sooner.

We measure this with cycle time. Cycle time is the amount of elapsed time that a work item spends as work-in-progress. In addition, business value that could have been realized sooner gets delayed because of too much WIP. This is known as cost of delay. It's a concept used to communicate value and urgency—a measure of the impact of time on the outcomes we want, such as customers buying our product this month instead of next month.

When you delay the delivery of a new feature because another request got bundled in with it, there's a cost for the delay of that new

feature. It could mean late feedback, less profit, or a missed sales lead opportunity. Your new feature gets hijacked on its way to your customer, and the more stuff you add, the longer the customer waits. If customers wait too long, they shop elsewhere. Once customers give up and move on, you lose. Maybe it was worth it—but do you really know?

In general, I define customers in two flavors:

External customers: People outside your organization who buy or use your product or service. If they move on to greener pastures, you lose revenue—and you risk a less-than-favorable review on your company’s Facebook or Amazon page.

Internal customers: People inside your organization who ask you to do something or who consume your work. A Product Development team is a customer of the security engineer who detects vulnerabilities in the product or the platform it sits on. An employee is a customer of the manager who provides feedback. Internal customers impact WIP. For example, help desk WIP grows when the accounting admin gets locked out of his computer. Marketing team WIP grows when the technical evangelist adds a new conference to their tour. Platform Operations WIP grows when the VP of whatever hires a third-party vendor to build a new integration.

WIP is a leading indicator of cycle time. The more items that are worked on at the same time, the more doors open up that allow dependencies and interruptions to creep in. Trailing or lagging indicators are backward focused—they measure performance data already captured.

Most metrics measured in technology and business, such as lead time (the elapsed time it takes to complete a request from the time it was first requested), cycle time, and throughput (the number of things completed over a period of time) are trailing indicators.

$$\text{LITTLE'S LAW}$$
$$\text{AVG CYCLE TIME} = \frac{\text{AVG WIP}}{\text{AVG THROUGHPUT}}$$

That is, we don't know how long certain things will take until those things are completed.

There is a relationship between the amount of WIP and cycle time—it's called Little's Law, where the average cycle time for finishing tasks is calculated as the ratio between WIP and throughput. WIP is a primary factor in the equation. It's obvious when you think about it—as soon as you get on a clogged freeway you know that your commute is going to take longer. For this reason, Thief Too Much WIP is the ringleader of all the other thieves.

You know Thief Too-Much-WIP is stealing time from you when:

Context switching is common: When computers context switch, the state of the process currently being executed is saved so that when it is rescheduled, the state can be restored to its correct spot. Because computers perform hundreds of context switches per second, it's easy to believe that multiple tasks are performed in parallel, when in reality the central processing unit (CPU) is actually alternating or rotating between tasks at high speed.

As Todd Watts writes in his blog post “Addressing the Detrimental Effects of Context Switching with DevOps,” the overhead incurred by a context switch, managing the process of storing and restoring the state, negatively impacts operating system (OS) and application performance.² Because a context switch can involve changing a large amount of data, it can be one of the most costly operations in an OS.³

Just like computers, humans incur overhead when context switching between different tasks. Although with humans, the overhead is much higher. Data structures containing all the information registers and OS-specific data, along with the exact point of entry for where to resume the process, aren't automatically rescheduled in the brain like they are in a CPU. Context switching in computers has a programmable flow to it.

The notion of flow in humans doesn't happen when context switching is the norm. Flow is the concept of focused motivation. It's characterized by complete absorption in what one does (energized focus). It's an optimal state that results in high levels of pro-

ductivity and satisfaction. To achieve flow is to be in the zone—that space where intrinsic motivation and creativity flourish.

To achieve flow, a focused concentration on the task at hand is necessary. This doesn't happen when distractions, whether in the form of email, food, coworkers, or social media, interrupt us. When we are responsible for multiple things, such as maintaining production and delivering new features, we shift what we're doing based on perceived priorities. By the time we get back to working on whatever we were doing before the interruption, we have to start all over again. Flow requires a “do not disturb” ethos.

Your customers wait for long periods of time: Flow also requires a level of efficiency. When it comes to flow efficiency, the length of time you keep your customer waiting is of prime consideration. If new projects are started before existing projects are finished, work piles up, requiring more resources and/or more people. It is inefficient from a customer perspective to prioritize starting new work over finishing the things you have already begun. If I'm writing a blog about kanban and the next step in the process is to have it edited by someone on the Marketing team, then beginning a new blog about DevOps before incorporating Marketing's edits for the kanban blog means I'll have to deal with a context switch when the editor gets back to me.

Quality suffers: Quality suffers from too much WIP. When I was at Corbis and took on the additional role of managing the new SAP basis team, I shot myself in the foot. I had to learn a complicated mainframe product while building a new team on top of continuing to do my original job. I hadn't done anything with mainframes since my first job out of college seventeen years before, and I didn't know anything about SAP. I didn't take the time to learn it very well because I had all these other things that had to be done on my plate. The result was a predictable one in retrospect: Neither the team nor SAP nor my other responsibilities got adequate attention. This led to a poorly managed team and an irritated me.

Irritated staff: Context switching is irritating—you’re rarely left with enough time to do a good job, nor with sufficient space to master the task or skill. Harry F. Harlow, an American psychologist, says in Daniel Pink’s book *Drive*, “The joy is in the pursuit more than the realization. In the end, mastery attracts precisely because mastery eludes.”⁴ Mastery eludes because there is insufficient time to pursue something long enough and deep enough before being interrupted.

Interruptions thwart deep thinking. Sherlock Holmes thinks best when he goes to his “mind palace” in the BBC adaptation of Conan Doyle’s famous sleuth’s escapades.⁵ Using a mental technique called the method of loci (Latin for location), he travels to his memory bank, a sort of mental map where memories are deposited, to withdraw memories. But he needs an environment void of distractions and interruptions, and he gets quite cranky if others interfere with his mind palace. And with good cause—it’s downright irritating to be interrupted when deep in thought.

Time thieves love the area of deep thought because, as David Rock relates in his book *Your Brain at Work: Strategies for Overcoming Distraction, Regaining Focus, and Working Smarter All Day Long*, it can take up to twenty minutes to get back to that same thinking spot after an interruption.⁶

Someone asks you if you have five minutes and you say yes: All too often, when someone asks you, “Do you have five minutes?” and you say yes, you end up working late. It’s annoying and sometimes exhausting, and we do it to ourselves. Even though I teach this stuff, I fall into this trap too. In our defense, we do get more endorphins from saying yes⁷—enough that even grouchy people want to say yes.

However, it’s not productive to say yes all the time, nor is it sustainable to consistently work evenings and weekends. The power of kanban gives you the freedom to finish work.

Kanban is Japanese for signal card—a card that, very simply, signals your availability to do some work. When you pull a card from the backlog onto the in-progress area of your kanban board, you commit to being available to do the work that the card represents.

The number of cards under the in-progress area reveals the amount of WIP on the kanban board. The board in Figure 2 shows a WIP of four (two items in prep, one in implement, and one in feedback). WIP limits are what make kanban a pull system. When a card is finished, it signals available capacity and causes another card to be pulled into the “In Progress” column.

Work flows across the board based on the WIP limits and pull policies. If WIP limits are set appropriately, the system cannot become overloaded. The WIP limit is what allows you to say, “No, there is no capacity to take on more work right now.” Think of reducing WIP not as limiting but as liberating. The right amount of WIP is what enables us to maintain a healthy amount of work.

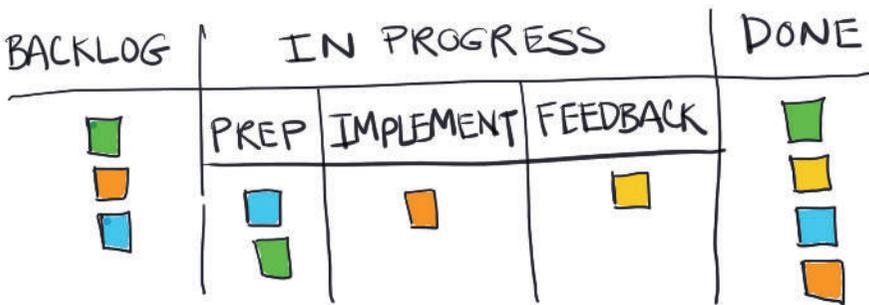


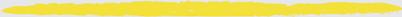
Figure 2: Prep Implement Feedback Board

When you say to your pal, “Yes, I’ll do that,” you have just authorized and prioritized “that” request over all the other requests in the backlog. Dan Weatbrook, Tableau WebOps Manager, calls this “born-in-doing”—a way of cutting in line, if you will.⁸ It’s a thief stealing time away from previous requests and is one reason why requests in the backlog take so long (and sometimes never make it) to the “In Progress” state.

These elements of too much WIP crop up across all the thieves. Thief Too Much WIP is the ringleader, and the other thieves steal ideas from this relentless troublemaker. We will get into more details on how the thieves interact with each other a bit later. For now, let’s take stock of

what we've covered so far about our thieving ringleader and move on to thief number two, Thief Unknown Dependencies.

KEY TAKEAWAYS



- We have a tendency to say yes to any request, regardless of how busy we are.
- Too much WIP prevents us from completing work on time, causes quality to suffer, increases costs, and irritates staff.
- Work-in-progress and cycle time have a relationship. High WIP means that other items sit idle, waiting for attention longer.
- Context switching, which wastes time, is a major consequence of too much WIP.
- We must learn to say no to additional work when our schedules are full.



1.2

The definition of freedom is that there is no dependency.

—Dada Bhagwan



UNKNOWN DEPENDENCIES

A friend of mine works for a company with \$23 billion in yearly revenue, where Product Team X deployed a component that broke Team Y's product. Now, Team Y's customers have to fork out \$5 million for the new Y part. This is on top of the \$10 million that they just paid to buy a new X part because the old X part was at the end of its life and was no longer supported. The customers in this scenario used parts created by both the X and Y teams. Part Y needed Part X to function correctly. The only way Team Y's customers could get their needs met was to buy the new X part.

Now this company has a major public relations disaster on their hands. They are losing significant market share because the two product teams didn't talk to each other. Team Y had zero visibility into Team X's decision to release a new version of software that Team Y's product was dependent on. The blame game, replete with finger pointing, begins and now a VP's head is on the chopping block.

It's very expensive when teams are unaware of mutually critical information. This is the type of thing that happens when there are unknown dependencies.

Let's define dependency. From my perspective, when we talk about dependencies, three types emerge:

1. **Architecture** (both software and hardware)—where a change in one area can break another area (i.e., cause it to stop functioning)

2. **Expertise**—where counsel or aid from a person with specific knowhow is needed to do something
3. **Activity**—where progress cannot be made until an activity is complete

If your manager is stuck in a meeting and thus you can't get the go-ahead to register for the conference before the day is out, then you've got yourself a dependency. Another example would be waiting on a test environment or a database restore from production before you are able to move your work forward.

Tightly coupled software architecture is a victim of the big bully that is Thief Unknown Dependencies. When a decision to remove a table from a database negatively impacts another team, Thief Unknown Dependencies scores big time. This is an example of a software code dependency.

Specialized expert skill sets are at risk of being hit by this big bully thief. A developer wonders, "Are there unknown vulnerabilities in this code?" while waiting on feedback from a security expert. But the security expert is busy discovering how someone hacked into their now unsecure database. A question waits on input from a database architect. "Is the data in the test environment wrong? Can they please check it out?" But the database architect is busy helping the security expert. When you are the only one on the team with a special skill set, you can be the bottleneck pulled in many directions. Expert skills in high demand are often unavailable when you need them. Thief Unknown Dependency snickers in delight.

A similar problem occurs for changes outside of your control in the form of third-party vendors. Major cloud providers, like Amazon EC2, Microsoft Azure, and Google Compute Engine, provide service level agreement policies that guarantee their customers 99.95% uptime. This equates to twenty-two minutes of allowable downtime per month. When your cloud provider is down, you are down, and Thief Unknown Dependency laughs at you. Granted, your cloud provider is a known dependency, but do you always know it when they first strike? How much time does the team spend troubleshooting a problem before they

realize it was the cloud provider who did it in the datacenter with the candlestick? But you still lose even if it's the provider's fault because you are constrained by contractual agreements. You may be compensated with time credits, but if a failure occurs, how much time is wiped out trying to recover lost data? If you totaled the hours a team troubleshoots an incident like this, how much time is really stolen?

Why Dependencies Matter

"Every dependency doubles your chance of being delayed or late."

— Troy Magennis

Troy Magennis gave an enlightening talk on dependencies at the Agile 2015 Conference in Washington, DC. Troy uses basic boolean logic (where all values are either true or false) to show that there is only ever one possible combination of inputs that result in an on-time delivery. Every time you remove one dependency, half of the total possible delay combinations are removed. If delivery requires every piece being complete, every dependency you remove doubles your chances of delivering on time.¹

Take a look at this example. If there are two inputs needed to deliver something, then there is only one chance in four of delivering on time. One chance in 2^n is the formula that computes the total number of binary permutations.

Come on now—math is fun! You got this. A binary digit can only be a 0 or a 1. A permutation is a way in which a number of things can be arranged. A binary permutation, then, is an arrangement of binary numbers. 2^n is 2 to the power of n . When the number of inputs is two, $n = 2$, and we have 2^2 , which equals 4, or 2×2 .

Let's write them all down to see how it works. There are four ways to have two inputs.

4 WAYS TO HAVE TWO INPUTS

0	0	→ 00	→ Everything is on time
	1	→ 01	→ LATE
1	0	→ 10	→ LATE
	1	→ 11	→ LATE

If there are three inputs needed to ship, then there is only a one in eight chance of on-time delivery.

8 WAYS TO HAVE THREE INPUTS

0	0	0	→ 000	→ Everything is on time
		1	→ 001	→ LATE
0	1	0	→ 010	→ LATE
		1	→ 011	→ LATE
1	0	0	→ 100	→ LATE
		1	→ 101	→ LATE
1	1	0	→ 110	→ LATE
		1	→ 111	→ LATE

Just by removing the strict delivery dependency you double the chances (from one in eight to one in four) of on-time delivery. There will always only be a single chance where everything is on time.

Imagine you have reservations for dinner with four people who travel independently to a fine dining restaurant, and you're told you won't be seated until all four people arrive. There are sixteen possible outcomes. That is to say, sixteen possible combinations for whether people arrive on time or not.

If you chart that out, fifteen outcomes have at least one person arriving late and only one outcome has everyone arriving on time.

Dependencies are asymmetrical in their impact. With four dependencies, it's not a 25% probability you won't be seated, it's a 93% (15/16th) probability that you won't be seated. There's a much greater chance that fifteen out of sixteen times, someone will be late. Time to call it a night and hit up the burger joint.

Figure 3, a three dependency chart, helps to visualize this math for three dependencies where the probability is 12.5% of being seated on time. When you add one more dependency, the probability is just one in sixteen, or .06% probability. Unless, that is, they work in Operations—then they'll never leave work early enough to arrive on time.

You know Thief Unknown Dependency is stealing time when:

- Coordination needs are high and project managers run around trying to get everyone aligned.
- People aren't available when you need them.
- A change in one part of the code/outline/plan unexpectedly changes something else.

YOU	FRIEND	BROTHER
Shaded	Shaded	Shaded

Figure 3. Three Dependency Chart

When the local pizza company delivers more than two pizzas to the same meeting room, pay attention. A two-pizza team is a team that can be fed with just two pizzas—about five to seven people depending on the size of the appetite. If three two-pizza teams need to have a joint meeting to discuss their dependencies on each other, then you have

high coordination costs. Fifteen to twenty-one people bantering their point of view can consume a lot of time. When was the last time fifteen people agreed on anything? When coordination needs are high, people aren't available when you need them to be.

Small teams can move fast. Nothing beats a small, cohesive group of people who communicate and collaborate effectively. The problem occurs when dependencies span across teams, causing things to break. When one team breaks another team's functionality by creating incompatible changes, the impacts can be destructive, as mentioned in the opening of this section about my friend's \$23 billion company. When we attempt to increase the performance of individual teams by breaking them down into smaller groups, hidden dangers await if there are unknown dependencies.

Cross-team communication is hard. When a bunch of two-pizza teams with lots of dependencies between them spend a lot of time coordinating to avoid stepping on each other's code (due to the merging of the different teams' code branches), the benefit of the small team diminishes. Smaller teams can increase integration costs. We like small teams because they can move fast. Just realize that by moving fast as an individual team, you may be paying the price of not moving very fast as a whole organization.

Lastly, remember the following common attributes from Thief Too Much WIP: costly context switching and expensive interruptions. Distraction is one of the biggest hurdles to high-quality knowledge work, costing almost one trillion dollars annually.

KEY TAKEAWAYS

- It's expensive when teams are unaware of mutually critical information.
- Architecture, expertise, and activities on hold are some of the common dependencies you might run into.
- Every dependency increases the probability you will be late. If possible, reduce dependencies to save time and money

and to avoid other complications. Conversely, every dependency you can find and eliminate doubles your chances of delivering on time.

- When coordination needs are high, people aren't available when you need them. The same is true for experts—when demand for a skill is high, experts are unavailable.
- When it comes to dependencies, individual team performance can increase to the detriment of company-wide performance.



1.3

Code will be used in ways we cannot anticipate, in ways it was never designed for, and for longer than it was ever intended.

—Joshua Corman



UNPLANNED WORK

Downtown, USA, Tuesday Morning

Picture this: A senior business executive sees business value in an integration between her company's product and another software application. She hires a third party to integrate a new service and promises zero impact to her Product Development teams.

The external, offshore team designs an integration but overlooks the fast growth of the user base, resulting in an overburdened database. The database server revolts, setting off alerts to the duty pager. Operations staff have to stop work on a high-priority task to troubleshoot the angry database. Two cups of coffee and two hours later, the issue is resolved, and people can get back to work on that high-priority task they were deep into before the interruption occurred . . . right after their meeting that starts in ten minutes. This was not the intent of the senior business executive.

People were interrupted and pulled away from important work. The interruption (the unplanned work) caught people off guard. It was an unintended consequence of a move by a well-meaning executive that negatively impacted high-priority work, work that will now be delayed because of the interruption. The time spent away from working on the original priority is irreversibly wiped out. This is the problem with unplanned work—it sets back planned work. It increases uncertainty in the system and makes the system less predictable as a result.

Sometimes unplanned work comes in the form of a necessary strategic change in direction: "Let's stop marketing to everyone and just

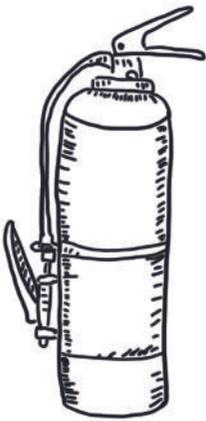
focus on large enterprises.” But often, unplanned work comes in the form of unnecessary rework or expedited work requests. These are the fires that stem from some failure. (Demand from failure is called, predictably, failure demand, and it is a frequent target for Thief Unplanned Work.) It can be the case, though, that a dependency from a team just down the hall from you is a greater risk to not responding to your needs. This usually results in a communication up your line of command to the common denominator leader and then back down the food chain to the person responsible, resulting in an interruption/delay to that person’s lunch (hopefully, it’s still in the fridge).

Let me be clear, I am not suggesting all work should be preplanned. It is irresponsible (maybe even delusional), to believe that it’s possible to know everything up front while planning a complex project. Quite the contrary—we don’t know much about what we don’t yet know. Sometimes changes in direction are necessary, because new information emerges as we work to solve problems. A major value of the Agile movement is to encourage responding to change over following a plan. Life is uncertain. Change is inevitable. It’s a law—the second law of thermodynamics to be precise.

Why Unplanned Work Matters

Unplanned and expedited work steals time away from work that’s creating value. The 2016 *State of DevOps Report* survey data show that high performers are able to spend 28% more time on planned work. Unplanned work is considered a measure of quality because the more unplanned work, the less time for creating value. “All hands on deck” incidents tend to reduce performance and increase variability.

As mentioned above, unplanned work steals time away from planned work. However, there are times when it is understandable and necessary for unplanned work to muscle its way to the front of the priority queue. If the request happens to be, “Please look into why no



one can log into the website,” then you really have no choice but to drop what you’re doing to fix the issue. Unpredictable fluctuations in demand can reduce the ability to deliver things as expected.

You know Thief Unplanned Work is stealing time from you when urgent issues pull people away from focused efforts on creating value. This could manifest in anything from an unexpected fire drill to a malfunction with a heavily used program that then adds uncertainty and variability into our everyday

work. Because of this interruption, something else is going to take longer than expected. If the work is frequently late, chances are Thief Unplanned Work (i.e., failure demand or strategic change in direction) is stealing not only your time but also your predictability.

The reality is we work in webs of interdependencies. The complexity of human interactions consistently produces things that no one wants. Thief Unplanned Work is a mainstay in a complex world where change and uncertainty flourish.

Unplanned work not only causes its own problems but brings with it all the problems of too much WIP: context switching, interruptions, delayed work, and increased cost. When unplanned work (such as fixing broken functionality on a website) creeps into the everyday work scene, it increases the amount of work already sitting on your plate. The more urgent unplanned work that interrupts your day, the bigger the pileup of partially completed planned work.

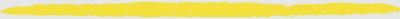
The relationship between unplanned work and too much WIP is a twisted, codependent liaison resulting in heaps of unexpected work that is impossible to catch up on. Unless you keep stepping up, planned responsibilities fall behind. Overfunctioning becomes an automatic response, eventually becoming dysfunctional and imbalanced. That’s why it is so important to learn to identify and tackle, as early and as often as possible, the issues created by Thief Unplanned Work.

Unplanned work increases risk and uncertainty, reduces predictability, and smacks down morale. But that doesn’t mean we have to simply lay there and let Thief Unplanned Work walk all over us. There are ways we can fight back.



Making work visible is a core component in combating thieves like Thief Unplanned Work and is also a core component of kanban, the system that we will come back to again and again throughout this book. Kanban cards show all kinds of information that traditionally have been hard to see. Kanban cards live on kanban boards, and kanban boards answer questions like, “What is being worked on?” and “What state is the work in?” and “Who’s doing what?” All essential information is visible on the board, so you don’t have to chase down workers to ask them what is happening, and you don’t have to wait for a politically sanitized weekly status report to get a glimpse of transparency.

KEY TAKEAWAYS



- Unplanned work adds unpredictability to the system.
- It’s all about predictability and expectations. Unplanned work eats expectations for breakfast.
- High-performing companies spend less time working on unplanned work than lower-performing companies.
- Sometimes we have no choice but to drop current projects to focus on urgent unplanned work.
- Unplanned work steals time away from planned work.
- Unplanned work is hard to see, but it can be made visible. Kanban helps to combat and better anticipate unplanned work by making work visible.
- Plan for unplanned work by reserving capacity for when it arrives.

