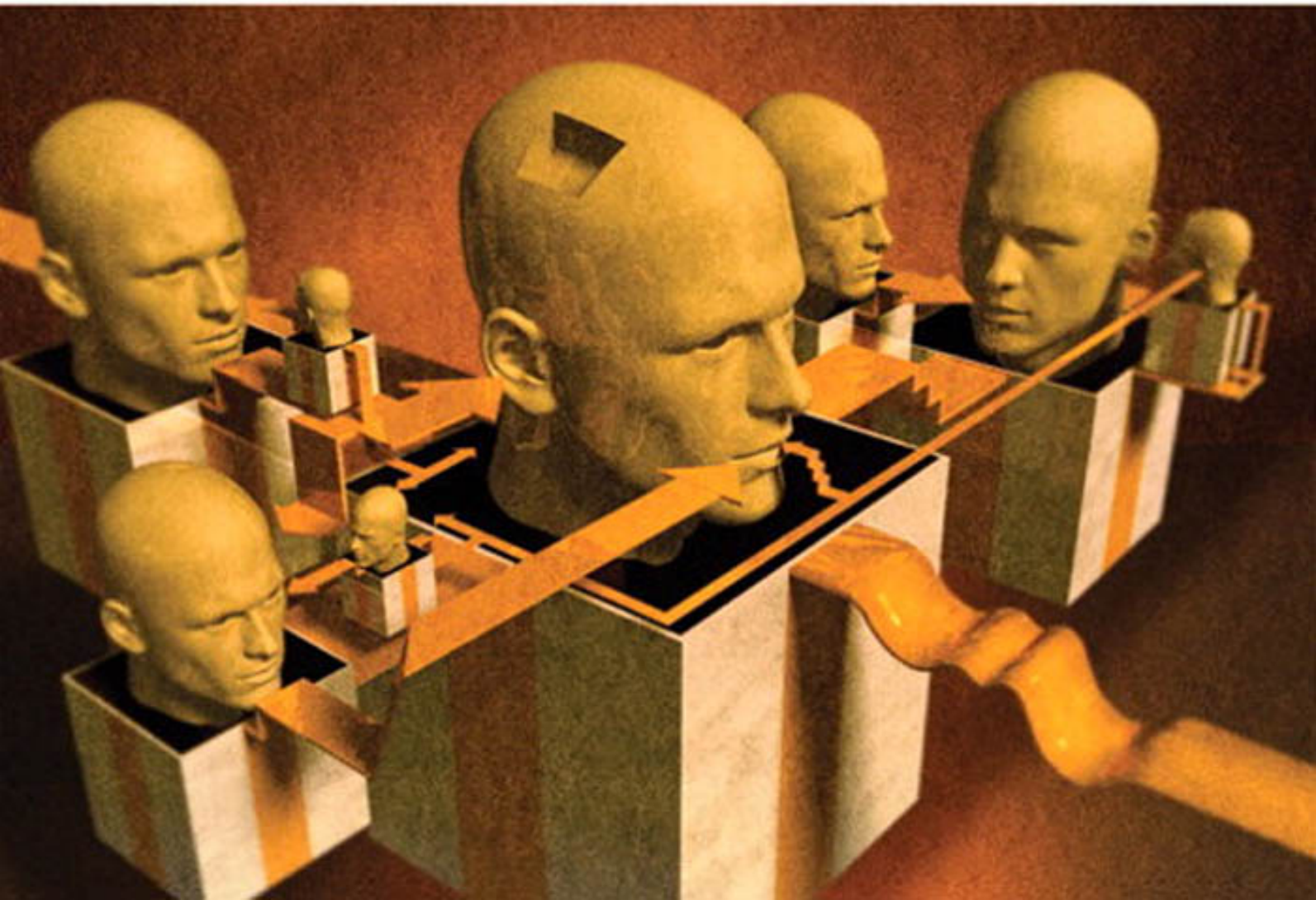
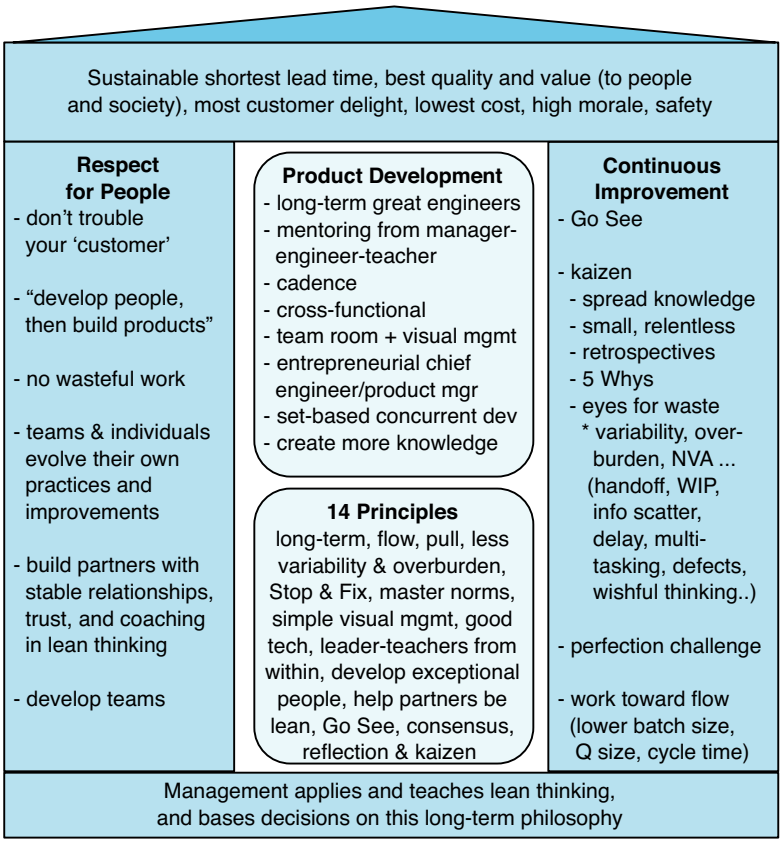
Scaling Lean & Agile Development

Thinking and Organizational Tools for Large-Scale Scrum

Craig Larman
Bas Vodde



THE LEAN THINKING HOUSE



Experiments

Systems Thinking

Lean Thinking

Queueing Theory

False Dichotomies

Be Agile

Feature Teams

Teams

Requirement Areas

Organization

- Try…Trial iteration 282
- Try…Lots of formal education and coaching 282
- Try…Lots of coaching 283

Large-Scale Scrum

- Try…Large-scale Scrum FW-1 for up to ten teams 291
- Try…Large-scale Scrum FW-2 for ‘many’ teams 298

Scrum Primer

- Try…Learn and do standard Scrum 308

Scaling Lean & Agile Development

This page intentionally left blank

Scaling Lean & Agile Development

Thinking and Organizational Tools for Large-Scale Scrum

Craig Larman
Bas Vodde

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data

Larman, Craig.
Scaling lean & agile development : thinking and organizational tools for large-scale Scrum / Craig Larman, Bas Vodde.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-321-48096-5 (pbk. : alk. paper) 1. Agile software development. 2. Scrum (Computer software development) I. Vodde, Bas. II. Title.

QA76.76.D47L394 2008
005.1—dc22

2008041701

ISBN-13: 978-0-321-48096-5
ISBN-10: 0-321-48096-1
Text printed in the United States at Courier in Westford, Massachusetts.
First printing, December 2008

To our clients, and my friend and co-author Bas

To 孙媛

This page intentionally left blank

CONTENTS

This page intentionally left blank

PREFACE

Thank you for reading this book! We've tried to make it useful. Some related articles and pointers are at www.craiglarman.com and www.odd-e.com. Please contact us for questions.

Typographic Conventions

Basic *point of emphasis* or *Book Title* or *minor new term*. A ***noticeable point of emphasis***. A **major new term** in a sentence. [Bob67] is a reference in the bibliography.

About the Authors

Craig Larman serves as chief scientist for Valtech, a consulting and outsourcing company with divisions in Europe, Asia, and North America. He spends most of his time working as a management and product-development consultant and coach within large or offshore groups adopting agile and lean product development, usually with an embedded systems focus. He led the agile offshore adoption (with Scrum) at Valtech India and served as creator and lead coach for the lean software development initiative at Xerox, in addition to consulting and coaching on large-scale agile and Scrum adoption for long periods at Nokia, Siemens, and NSN, among many other groups. Originally from Canada, he has lived off and on in India since 1978. Craig is the author of *Agile and Iterative Development: A Manager's Guide* and *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design & Iterative Development*, the world's best-selling books on agile methods, OOA/D and iterative development. Along with Bas, he is also co-author of the companion book *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*.

After a failed career as a wandering street musician, he built systems in APL and 4GLs in the 1970s. Starting in the early 1980s he became interested in artificial intelligence (having little of his own). Craig has a B.S. and M.S. in computer science from beautiful Simon Fraser University in Vancouver, Canada.

Bas Vodde works as an independent product-development consultant and large-scale Scrum coach. For several years he led the agile

and Scrum enterprise-wide adoption initiative at Nokia Networks. He has been a member of the leadership team of a very large multisite product group (in Europe and China) adopting Scrum. Bas has worked as a senior developer/architect in embedded telecommunication systems, in addition to serving as a quality manager. He has led the development of solutions and the coaching for test-driven development in embedded systems. Along with Craig, Bas is co-author of the companion book *Practices for Scaling Lean & Agile Development.* Originally from Holland, he has lived in China for years and is now based in Singapore.

Acknowledgments

Thanks to all our clients.

Thanks to reviewers or contributors, including Peter Alfvin, Alan Atlas, Gabrielle Benefield, Bjarte Bogsnes, Mike Bria, Larry Cai, Mike Cohn, Pete Deemer, Esther Derby, Jutta Eckstein, Kenji Hiranabe, Clinton Keith, Kuroiwa-san, Diana Larsen, Timo Leppänen, Eric Lindley, Mary Poppendieck, Tom Poppendieck, Ken Schwaber, Maarten Smeets, Jeff Sutherland, Dave Thomas, and Ville Valtonen.

Current and past Flexible company team members (and reviewers), including Kati Vilki, Petri Haapio, Lasse Koskela, Paul Nagy, Joonas Reynders, Gabor Gunyho, Sami Lilja, and Ari Tikka. Current and past IPA LT members (and reviewers), especially Tero Peltola and Lü Yi.

Bas appreciates his wife Sun Yuan for all the time he had to "focus on the books" rather than on each other, and for her support when moving or traveling to different countries to work with different product groups. And he thanks Craig for the stimulating discussions and the years working together with large products and with debugging organizations—and Bas's writing.

Craig thanks Albertina for her help so that he could write.

Thanks to Louisa Adair, Raina Chrobak, Chris Guzikowski, Julie Nahil, and Mary Lou Nohr for publication support.

This page intentionally left blank

Chapter

Book

Chapter

1

INTRODUCTION

The future ain't what it used to be.
—Yogi Berra

We sat down in the meeting room with our hot coffee. Outside was a bitter-cold north European winter morning. In came our new client and we shook hands. "Thanks for visiting," he said. "First, you should know that our product group is not large, maybe only eighty developers."

We once met a group adopting agile development that was not sure if they could grow to *very* large-scale development: 12 people.

People have different scales in mind regarding 'large.' To some it means only 50 people or even less. To others, much more. We define a large product[1] group as *one whose members' names you could not remember if you were all together in a room.* We work typically with single-product groups in the range of 100–500 people that are adopting Scrum, lean principles, and agile development practices, usually on software-intensive embedded systems. So by this definition—at least with our limited memories—this is the realm of 'large.'

On to our key recommendation: After working for some years in the domains of *large*, *multisite*, and *offshore* development, we have distilled our experience and advice down to the following: *Don't do it*.

There are better ways to build large systems than with many developers in many places. Rather, build a small group of great developers and other talents that can work together in teams, pay them well, and keep them together in one place with product management or whoever acts as the voice of the customer.

1. Scrum (and this book) applies both to product development for an external market, and to internal applications (internal products).

But of course you are *still* going to do large, multisite, or offshore development. This is because your existing system is already structured that way, or because—in the case of large groups—there is the mindset that "big systems need lots of people." We regularly coach groups that ask, "How can we calculate how many people we will need?" Our suggestion is, "Start with a small group of great people, and only grow when it really starts to hurt." That rarely happens.

Since large, multisite, and offshore development is going to happen, we would like to share what we have tried or seen at the intersection of these domains with lean and agile product development principles and practices.[2]

Thinking and Organizational Tools

When Bas was a member of the leadership team of a large product group, he frequently (in meetings) asked, "Why do we have this policy? … What will happen to the organizational system if we do that?" Months later a member of the team told Craig, "It drove me nuts to keep hearing those questions. But later, I appreciated it." Bas wasn't trying to be annoying; he was trying to suggest and encourage *systems thinking*—a thinking tool (1) to consider the deeper dynamics of the development system as a whole, (2) to understand how a system became the way it is, and (3) to reconsider assumptions underlying the existing organization.

When people are introduced to Scrum with its short timeboxed development iterations, they first see it as a localized practice to incrementally grow a product in small manageable steps, with learning and corrective actions toward a goal. Consequently, people will say, "Oh, 'agile' doesn't affect me; that's a *development* practice." But there is a bigger picture and a potential higher-level learning

2. The companion book is *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. It covers detailed practice tips related to scaling and planning, product management, multisite, offshore development, contracts, requirements, design and architecture, coordination, legacy code, testing, and more.

loop beyond the lower-level development learning cycle: a learning organization of people that repeatedly re-examine the structures and policies that define and surround agile product development. The result of adopting Scrum or lean principles in very large product groups inevitably leads to this higher-level organizational learning challenge.

Example: Consider an enterprise whose R&D division tries to be more adaptive by adopting Scrum. The Sales division continues in their old mode: Maximize personal commissions and quarterly sales by promising the moon and the stars to customers, combined with almost boundless optimism for what "our great people in R&D can do." Faced with unattainable 'commitments' R&D did not themselves design or make, R&D is then blamed for not meeting "*our* promises," and it is concluded that "Scrum doesn't really help."

If this were a book about adopting Scrum only in one small 20-person single-product group within a large enterprise, systems thinking and organizational tools would be interesting but non-vital topics. But they are vital to a successful adoption when Scrum is being scaled to a 400-person single-product group, probably within a larger R&D organization in the thousands that is also making the transition, with deep connections to the Sales and Delivery groups, and constrained by traditional Human Resource and Enterprise Governance policies on team structures, reporting, measurement, milestones, contracts, and rewards.

Consequently, this book suggests that one cornerstone for large-scale Scrum and agile development is people who learn and apply various *thinking tools*, including (but not limited to) systems thinking, mental-model awareness, lean thinking, queueing theory, and recognition of false dichotomies.[3]

With those thinking tools in place, it will become increasingly clear that the existing organizational design inhibits flow of value, leading to pressure for redesign. Hence, this book suggests a second cornerstone of *organizational tools*, including feature teams,

3. The term *thinking tools* was popularized in [Reinertsen97].

requirement areas, and many other changes in structure, process, task, people, and rewards.

Action Tools

In parallel with adopting thinking and organizational tools, many *action tools*—specific development practices—help the product group get going on large, multisite, and offshore agile development. The *effective* use of these action tools—shared in the companion *Practices* book—is somewhat dependent on organization redesign. Many practices can be tried without deeper structural change, but constraints on benefit will be felt.

So the tools in this book could be seen as prerequisites for the actions tools of the companion book. Yet in reality, *practices* will be adopted first—because that is where people want to start. And that will eventually invite a look back at thinking and organizational tools.

We suggest that coaches and other change agents involved in the adoption of large-scale Scrum or lean development acquaint themselves early with thinking and organizational tools, while in parallel helping to introduce action tools. At some point the situation will be ripe—people will be ready—for a turn in the discussion from "*How do we do large-scale continuous integration*?" to "*Do existing HR policies prevent real teams*?" and "*What is flow of value and what inhibits it in our organizational design*?"

Experiments: Try… and Avoid…

Scrum emphasizes empirical process control; there is too much complexity and variability for a cookbook approach to processes for development. Therefore, the tools in both books are presented as a series of *tips* that start with *Try…* or *Avoid…* to suggest experiments, nothing more. They certainly may not work in your circumstance. The approach both in Scrum and in the lean thinking practice of *kaizen* is to first *inspect* and grasp the existing situation. Then, second, to *adapt* with new improvement experiments. The

attitude of endless experimentation is vigorously encouraged in lean thinking; perhaps the only bad process-improvement experiment is the one not tried. At Toyota, Taiichi Ohno—arguably the key contributor to lean thinking—would visit an area and inspect any written standards document. If it was covered with dust or otherwise not recently changed, he would grow quite impassioned and urge people to always evolve their 'standards.'

In Scrum this inspect-and-adapt (experiment) cycle repeats every two- or four-week timeboxed iteration as long as the product exists. And in lean thinking, this continuous experimentation and improvement cycle applies both to individual products and *to the enterprise as a whole*.

LIMITATIONS

There is still much for us to learn about these domains. What we have written here and in the companion book reflects our current (limited) experience and understanding, which we hope will evolve in the coming years. For example, although we have lived for some years in China and India, we feel we have barely scratched the surface in terms of our multicultural experience and insight in relation to offshore and multisite agile development. Nevertheless, our sincere wish is that these tips are of some value to you. We welcome further insights and stories from our readers.

Large-scale Scrum can influence almost all aspects of a product-centric enterprise. To keep the scope of this material manageable and because of our limited experience in some of these areas, we bounded or deferred subjects that are worthy of more discussion. These include:

- budgeting and finance
- sales
- marketing
- hardware development
- product development not involving any software
- deployment/delivery
- field support

Essentially, this book is relevant to general-purpose product development. Scrum and lean product development are not limited to

software systems [NT86]. However, the bias is toward software-intensive systems (usually embedded) because of our background and because of the ever-growing ubiquity of software in everyday devices, from washing machines to shoes.

Especially in this book we dissect some assumptions and policies in traditional organizations that inhibit *flow of value* and *effective teams*. This analysis may come across as startling or challenging at times. We do not mean to give offense, but organizational redesign to support lean and agile development will not happen without increased scrutiny of traditional assumptions and increased transparency. Organizational change can also lead to displacement of talented people from old roles. As in Toyota, we encourage finding new areas of contribution for people within a company—both because skilled people deserve this, and because otherwise it inhibits change.

With both books combined pushing over 700 pages, we regret that we could not write or think better to make the subject of *large*…smaller.

On to thinking tools…

Thinking Tools

Chapter

Book

Chapter

2

SYSTEMS THINKING

I took a speed reading course and read "War and Peace" in twenty minutes. It involves Russia.
—Woody Allen

"No matter what we do, the number of defects in our backlog remains about the same," a manager told us; this for a 15 MSLOC C and C++ product with several hundred developers where we were working (and adopting lean principles). What's going on? Systems thinking may help. In small groups the forces at play are more quickly seen and informally understood, but in large product development—or any large system—it's tough. Gerry Weinberg highlights two decisive factors in this situation:

> *Weinberg-Brooks' Law: More software projects have gone awry from management's taking action based on* ***incorrect system models*** *than for all other causes combined.*
>
> *Causation Fallacy: Every effect has a cause… and we can tell which is which. [Weinberg92]*

These reflect the impact of our **mental models** on the system, a subject that will be revisited later in the chapter.

Problems stemming from mental models and assumptions are one issue. Another is that large-scale adoption of Scrum, lean thinking, and agile principles is not isolated to the development group. It bumps into product management, budgeting, beta-testing, launch, and governance and HR policies. Accordingly, in large-scale agile adoption it is useful to be able to get together with colleagues and *effectively reason* about the mental models, causal relations, feedback loops, and control mechanisms (or illusions of control) in a big system that is about to be seriously *perturbed.* Systems thinking is one of those reasoning tools.

It Depends on Common Sense?

"*It depends on common sense.*"—A statement sometimes heard in Scrum and common parlance. But what is this? Einstein quipped, "*Common sense is the collection of prejudices acquired by the age eighteen.*"

Taiichi Ohno, the father of the Toyota Production System, said, "*[…] misconceptions easily turn into common sense. When that happens, the debate can become endless. Or, each side tried to be more outspoken than the other and things do not move ahead at all. That is why there was a time when I was constantly telling people to take a step outside of common sense and think by 'going beyond common sense.' Within common sense, there are things that we think are correct because of our misconceptions. Also, perhaps a big reason we do some of the general common sense things we do is that based on long years of experience, we see there are no big advantages to doing things a certain way but neither are there many disadvantages to it. … we are all human so we're like walking misconceptions believing that the way we do things now is the best way. Or perhaps you do not think it is the best way, but you are working within the common sense that 'We can't help it, this is how things are*'" [Ohno07].

"Common sense" is not so reliable when trying to understanding nonlinear systems—such as large-scale product development.

In 1958, the *Harvard Business Review* published "Industrial Dynamics: A Major Breakthrough for Decision Makers," a landmark paper by Jay Forrester, MIT Sloan School professor [Forrester58]. This paper spurred the movement of systems thinking in business education, and the MIT Sloan School of Management became known for educating people in **system dynamics**. System dynamics is sometimes treated as a synonym for **systems thinking**, though the latter is a more general term.

MIT also attracted other system-dynamics-oriented researchers such as Peter Senge.[1]

1. Senge wrote *The Fifth Discipline*, on systems thinking and learning organizations, named "one of the seminal management books of the last 75 years" by the *Harvard Business Review.* See [Senge94].

Consistent with *Weinberg-Brook's Law*, Forrester's research showed that decision makers who were given dynamic models of a business system and asked to improve their output performance, *usually made them run worse* [SKRRS94]. The observation was that most people have weak judgement on how to fundamentally improve systems, usually applying incorrect "common sense" and quick-fix 'solutions' that do not create long-lasting systemic improvement.

Why is the behavior of a large development group (a system) not understood or guided skillfully? The answer lies, in part, in the behavior of stochastic systems with queues and variability, as explored in the Q*ueueing Theory* chapter. And the same answer lies in *control theory*: Most systems of interest—such as a product development group—have complex positive and negative feedback loops and nonlinear behavior. The behavior of these systems defies our gut instinct. And then there is the minor issue of *people*.

In summary, reasons for not being skillful in fathoming or guiding a big system include (but are not limited to):

- lack of knowledge about the system dynamics, feedback loops, nonlinear systems behavior, and unintended consequences in workplace systems
- not understanding root causes of problems (and how to find)
  - *causes,* not cause; in systems thinking one sees that there are multiple, indirect, and dynamic causes to problems
- not knowing if or why quick-fix or local-department decisions degraded overall delivery performance.

In short, not being systems thinkers.[2]

These reasons are consequential at the intersection of management and large-scale adoption of lean and agile principles. The leadership team is part of the system being perturbed; if they do not apply systems thinking, they could *really* perturb it—and not in a good way.

2. Another reason: Believing more control is possible than actually is. Complexity science suggests fundamental limits on predicting and controlling semi-chaotic social systems [Stacey07]. This is a rather large can of worms that will remain unopened in this book.

As a summary of systems thinking insight, we like the 'laws' described in *The Fifth Discipline*:

- Today's problems come from yesterday's 'solutions.'
- The harder you push, the harder the system pushes back.
- Behavior will grow worse before it grows better.
- The easy way out usually leads back in.
- The cure can be worse than the disease.
- Faster is slower.
- Cause and effect are not closely related in time and space.
- Small changes can produce big results…but the areas of highest leverage are often the least obvious.
- You can have your cake and eat it too—but not all at once.
- Dividing an elephant in half does not produce two small elephants.
- There is no blame.

Toyota's internal motto is "Good thinking, good products." Systems thinking is a set of *thinking* tools to help…

- ❑ **see system dynamics**—a development organization is a system of people and policies with subtle feedback loops and unintended consequences
  - – we can learn to see and thus improve the system with **causal loop diagrams** created in a workshop
- ❑ **see mental models**—one reason behind suboptimal decisions is mistaken assumptions and faulty reasoning
  - – casual loop diagramming and Five Whys expose these
- ❑ **see root causes**—real improvement requires learning how to find root causes of problems and see deeper relationships
  - – causal loop diagrams, 5 Whys, and Ishikawa diagrams reveal these
- ❑ **see local optimization**—another source of suboptimal decisions is **local optimization**, making the 'best' decision from the viewpoint of a person or department, rather than **global optimization** for the lean systems-level goal of *deliver value fast with high quality and high morale*.

This chapter is organized around the following areas in systems thinking: Learning to see (1) *system dynamics*, (2) *mental models*, (3) *root causes*, and (4) *local optimization*.

SEEING SYSTEM DYNAMICS

Static versus Dynamic Complexity

Many of us, especially in engineering and finance, are educated to master **complexity of static details**—learning to analyze and manage information (requirements, financial analysis, …), decompose complex structures into simpler ones, and so forth. That is, complexity of a static, information, or structural nature.

Why do big software systems tend to degrade, with more and more time spent on defects? What might happen if the USA invades Iraq? Seeing the dynamics behind these questions involves analysis of the **complexity of dynamics**.

In contrast to static-details education, many of us receive no *formal* education in analyzing *dynamics* complexity[3], especially workplace dynamics. Perhaps there is a belief it is sufficient to rely on common sense in the workplace. Forrester demonstrated that "common sense" is just not so in complex systems, and showed it is possible to formally educate people to become better system dynamics thinkers in the workplace using *dynamic system models* visualized in *flow diagrams* [Forrester61].

Flow diagrams encompass material, financial, and information flows, stocks (variables with a quantity, such as cash or number of defects), the impact of decisions and policies, and cause-effect relations. A popular simplification is the **causal loop diagram** that focuses on cause-effect relationships and feedback loops in a system [Sterman00]. There are a variety of similar notations; they all show stocks (variables), causal links, and delay. In [Weinberg92] this is called the *diagram of effect*.

3. Macroeconomics, psychology, sociology, and biology are exceptions, among many others.

The First Law of Diagramming: Model to Have a Conversation

A tool to learn to see system dynamics is a causal loop diagram, ideally sketched on a whiteboard in a Sprint Retrospective with colleagues. Before going further, here is the *First Law of Diagramming*

> *The primary value in diagrams is in the discussion while diagramming—we model to have a conversation.*

When a group gets together to sketch a causal loop diagram on a whiteboard (Figure 2.1), the primary value is the conversation and shared understanding they arrive at while creating the model. Its visualization as an easy-to-see diagram *is* important to make concrete and unambiguous (on the whiteboard) the ideas—the mental models people have—because words alone can be fuzzy and misunderstood. But still, the diagram is secondary to what people take away: learning and a revised understanding through a discussion.

Figure 2.1 it is the the acts of discussing and thinking that are most important when diagramming, Valtech India

Basic Problems and Simple Enjoyable Tools

Over the years, we have learned sophisticated analysis and design skills and heuristics for engineering, management, and more. At first we were inspired and excited to apply and share all these, until we realized in the course of real-world work…

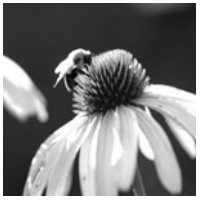
> *The vast majority of problems in business (including development) are so basic that a key solution is education in and consistent use of* ***simple, enjoyable*** *thinking and action tools.*[4]

Simple—For example, system dynamics and causal loop modeling books and courses can get overly complicated, with unnecessary overhead such as the *archetypes* idea described in the *Fifth Discipline*, computer simulation, nonlinear equations, and so forth. In practice, this serves to intimidate ordinary people from experimenting with—and sticking with—what in essence can be applied as a simple tool: standing around a whiteboard to sketch, discuss, and model basic cause-effect dynamics in business.

When considering any thinking or action tools for the workplace reality, know that

> *Le mieux est l'ennemi du bien. (The best is the enemy of the good.)—François-Marie Arouet (Voltaire)*

Enjoyable—There are many intricate thinking or action tools that professors or methodologists bemoan are not used—or at least not sustainably used. Why, on the other hand, are the practices in Scrum or Extreme Programming (XP) often adopted and *remain sticky* in practice? First, there is quick value to the hands-on worker participants—the cost/benefit ratio is attractive and pays fast. Second, they are not painful; some will even say they are interesting or enjoyable. It is not uncommon for people in a system dynamics sketching workshop to say it was interesting (and useful). Humans are humans; enjoyable practices are important for sustainable use.



Emphasis on such tools is especially important when scaling to large product development, because the ability to *push* practices and processes grows very weak as group size increases. As a bee is attracted to colorful fragrant flowers, you want to *attract* people to simple, enjoyable tools, including…

4. 'Basic' does not mean trivial or easy to solve. For example, 'motivation' and 'quality' are basic but not easy issues.

Try…Causal loop sketching workshop to see system dynamics

Causal Loop Diagrams

Causal loop diagrams are presented several times in this book, to help see the dynamics of what is going on in large-scale development. It is useful to understand them for that reason alone. And more useful to you, we recommend:

Try…Sketch causal loop diagrams at whiteboards with others

The practical aspect of this tip is more important than may first be appreciated. It is vague and low-impact to suggest "be a systems thinker." But if you and four colleagues get into the habit of standing together at a large whiteboard, sketching causal loop diagrams together, then there is a concrete and potentially high-impact practice that connects "*be* a systems thinker" with "*do* systems thinking."

The following examples seem sterile when presented in a book. But imagine you were at a whiteboard with other people and the diagrams were being sketched during a lively conversation. That's the way we suggest 'doing' systems thinking.

Concrete modeling tip: We start by writing on sticky notes to define *variables*. A note might read "feature velocity" or "# defects." We place these on a whiteboard. Then we sketch causal link lines between the sticky notes. There will be (or should be) lots of rewriting, erasing, and redrawing during the modeling session. The most meaningful outcome is *understanding*; in addition, some participants will want to take a digital photo of the whiteboard sketch.

Notation and Examples

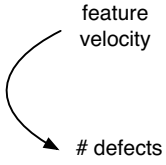
Causal loop diagrams contain many elements; the following common useful subset is explored through a scenario.

- variables
- causal links
- opposite effects
- constraints
- goals
- reactions; quick-fix reactions
- interaction effects
- extreme effects
- delays
- positive feedback loops

The following simplified scenario is for a particular organization. It is not a generalization.

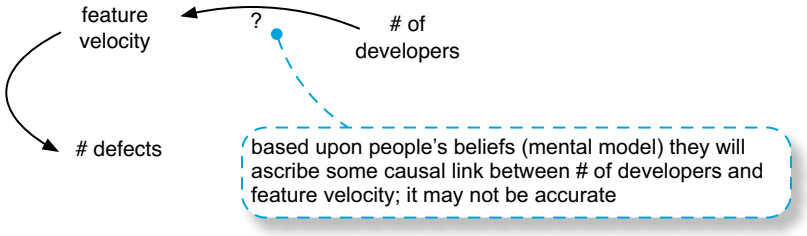
Variables—Causal loop diagrams include *variables* (or stocks) such as the *velocity (rate of delivery) of software features* and *number of defects*. Variables have a measurable quantity.

Causal links—An element can have an effect on another, such as if feature velocity increases, then the number of defects increase; that is, more new code, more defects.



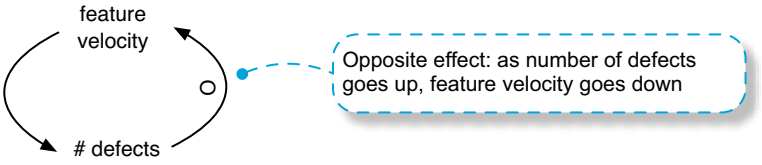
Now it is time to bump into *Weinberg-Brook's Law* and the *Causation Fallacy*. It is easy to sketch a diagram; it is something else to model with insight. For example, consider the relationship between the *number of developers* and *feature velocity*.

The nature of any cause-effect relationship is actually not obvious, though it is common for people to jump to conclusions such as more developers means better velocity. Adding people late in development may *reduce* velocity (a sub-element of "Brooks' Law" [Brooks95]). Or, *more* bad programmers could really slow you down. An argument can be made that *removing* terrible developers can *improve* velocity.



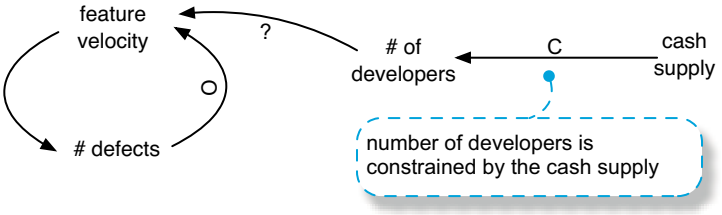
Opposite effects—A causal link effect may be the same or opposite direction; if A goes up then B goes up, or vice versa. Opposite effect

is shown with an 'O' on the line. Suppose defects going up puts a drag on the system, lowering the velocity of new features because people spend more time fixing or working around bugs.

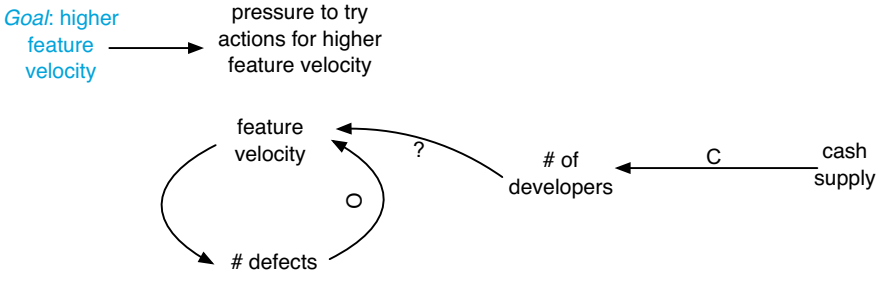


Constraints—Unless you can find people to work for free, there is a constraint on the number of developers, based upon cash supply.

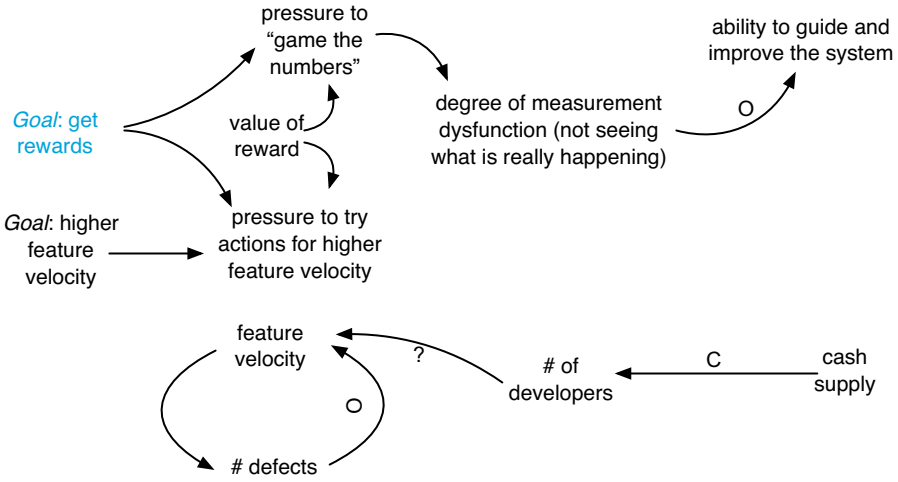
Constraints are *not* causal links. As cash supply goes up, it is not the case that the number of developers goes up.



Goals and Reactions—People, departments, and systems have goals, such as *higher feature velocity.* Goals often generate pressure for people to react (or act), with the intent of achieving the goal. But since there is *Causation Fallacy* and *Weinberg-Brooks' Law* to contend with, people should be cautious about assuming what actions will help. Now a goal and pressure for reaction is shown:



Not only does a goal with a *reward* create pressure to act, but also it creates pressure to *appear* to be acting and achieving, due to the **measurement dysfunction** generated by rewards. And the measurement dysfunction can be proportional to the perceived value of the reward because people are being motivated to get a reward, not to improve the system [Austin96]. Notice how rewards can actually degrade system performance. Visually, the system dynamics may be…

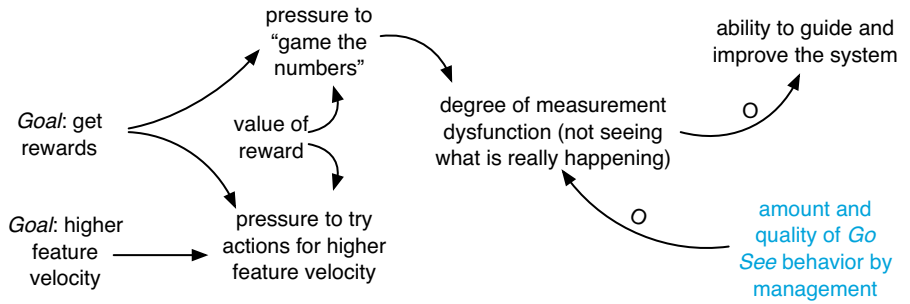


It is quite interesting that all these dynamics have been added by introduction of reward, and yet there is no necessary connection between the top part of this model and the bottom.

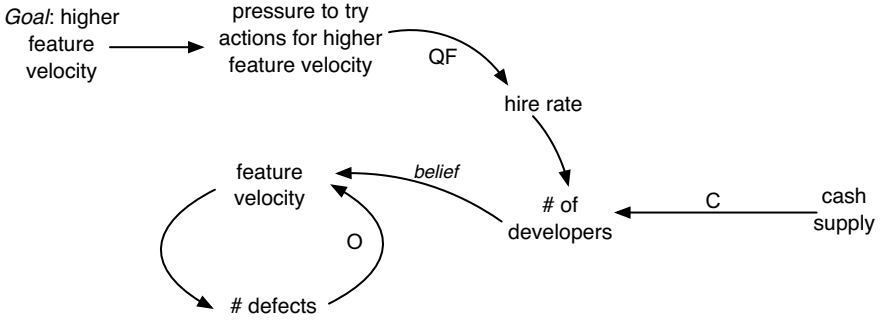
There is no guarantee that feature velocity has improved—or even been worked on.

Removing the reward system is a root-cause solution to the dysfunction. Another (lesser) surface countermeasure is the lean *Go See* principle and management behavior:

Go See p. 52



Quick-fix reactions—One difficult and slow solution toward the goal of higher velocity is to hire great developers, to increase coaching and education of existing staff, and to remove terrible workers. The alternative is called a *quick fix*, a reaction that is hoped to achieve the goal quickly and with less effort. Sometimes a quick fix works well both in the short and long term, really strengthening the system. Sometimes not…hence, "faster is slower." For example, people may *believe* that increasing the number of developers increases the feature velocity. And they may thereby hope that hiring more developers will most quickly and easily solve the velocity problem. 'QF' indicates the quick fix:



Interaction effects—There is the constraint of cash supply on hiring. One hard and slow solution is to get more cash. A quicker fix is to hire *much* cheaper developers. In this case, the level of cash supply now has an *interaction effect* with other causal links. Low cash