Net Objectives
ESSENTIAL WHITE PAPERS

Becoming Lean: Why, What, and How

by Al Shalloway

Becoming Lean: Why, What, and How

by Al Shalloway

A Net Objectives Essential White Paper

Net Objectives Press, a division of Net Objectives

1037 NE 65th Street Suite #362

Seattle, WA 98115-6655

404-593-8375

Find us on the Web at: *www.netobjectives.com*

To report errors, please send a note to *info@netobjectives.com*

Copyright © Net Objectives, Inc. All Rights Reserved.

Published by Net Objectives, Inc.

Net Objectives and the Net Objectives logo are registered trademark of Net Objectives, Inc.

Notice of Rights

No part of this publication may be reproduced, or stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise without the written consent of Net Objectives, Inc.

Notice of Liabilities

The information in this book is distributed on an "As Is" basis without warranty. While every precaution has been taken in the preparation of this book, neither the authors nor Net Objectives shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer or hardware products described in it.

10 9 8 7 6 5 4 3 2

Printed in the United States of America

BECOMING LEAN: THE WHY, WHAT AND HOW

Many companies have heard that the concepts and methods of Lean would be of use to their organization; however, they do not see how something that sprang from manufacturing practices could apply to software development. This article presents a different way of looking at Lean Software Development – one that is independent of Lean's manufacturing heritage. It begins by presenting Lean as a collection of a body of knowledge applying Lean principles to software development. It then shows how this creates a new paradigm of management, one that does not inevitably lead to micro-management or chaos. Finally, it concludes with a discussion about how organizations can use Lean to improve their ability to learn.

THE LEAN PARADIGM SHIFT – LEAN SCIENCE

Lean represents a paradigm shift from focusing on increasing productivity to focusing on shortening the time from the beginning of work to the completion of it. While Lean adopters are looking for higher productivity and lower cost, they have learned that going after these directly actually results in lower true productivity (value delivered per person) and higher costs. The reason is that productivity measures too often are geared toward how much work people are doing rather than how much true value is being delivered and cost, alone, is inadequate for deciding on process and/or product improvements. For example, measuring how much work people are doing leads to keeping people busy. Unfortunately, this leads to overworked analysts, developers and testers are incredibly busy while seemingly taking forever to deliver what the business stakeholders need. It does not translate into true added value.

Lean science could be said to be the set of testable knowledge based on the foundations of Lean thinking. To summarize, these foundations are

- Attend to the system in which development takes place; that is where the bulk of errors are;
- Respect the people doing the work;
- Attend to the time from when work starts until it is consumed by the customer ("cycle time");
- Get to the root cause of errors when they occur.

The foundational tenet of Lean is, "focus on shortening cycle time." Productivity will follow. In other words, Lean is based on the hypothesis that shortening cycle time raises productivity by reducing the delays that make error detection more difficult and it lowers cost because this delay in error detection increases the amount of work needing to take place.

Consider the type of work we do in software development, as illustrated in Table 1.

These items are not listed in any particular order; I am not implying any particular process. The activities on the left represent work that provides real value while those on the right represent work which we often must do but which don't really

Alan Shalloway is the founder and CEO of Net Objectives. With over 40 years of experience, Alan is an industry thought leader in Lean, Kanban, product portfolio management, Scrum and agile design. He helps companies transition to Lean and Agile methods enterprise-wide as well teaches courses in these areas. Alan has developed training and coaching methods for Lean-Agile that have helped Net Objectives' clients achieve long-term, sustainable productivity gains. He is a popular speaker at prestigious conferences worldwide. He is the primary author of *Design Patterns Explained: A New Perspective on Object-Oriented Design, Lean-Agile Pocket Guide for Scrum Teams, Lean-Agile Software Development: Achieving Enterprise Agility and Essential Skills for the Agile Developer*. Alan has worked in literally dozens of industries over his career. He is a co-founder and board member for the Lean Software and Systems Consortium. He has a Masters in Computer Science from M.I.T. and a Masters in Mathematics from Emory University. You can follow Alan on twitter @ alshalloway

add value. In the Lean world, these latter activities are called "waste." And it is good to try to eliminate waste.

Activities that provide real value	Activities that we do but provide no value
• Getting requirements • Planning • Design • Collaboration • Programming • Testing • Integration • Deployment • Documentation • Training	• Re-doing requirements • Working from old requirements • "Fixing" bugs • "Integration" errors • Building unneeded features • Overbuilding frameworks

Table 1: Activities performed during software development

The problem with trying to eliminate the waste is it is often created by people who are not doing the work. Consider the problem of two ditch diggers. Both are working hard. One is digging away and throwing his dirt into the other guy's hole. Stepping back, you can see the first guy is creating extra work – "waste" - but down in the trenches, the second guy just sees still more dirt that he has to remove. Sometimes waste can't be seen until you look at the entire picture.

It is easier to say "eliminate waste" than it is to actually eliminate it. We can't just stop doing wasteful things; we must stop the need for doing them. Let's consider the wasteful activities on the right-hand side of Table 1 and what is driving the need for them. It will become clear that the root cause involves delays, and that is what we want to eliminate.

Re-doing requirements is a common practice in software development. It occurs because we often get requirements too early. There is a delay between when a requirement is first mentioned until the development team starts working on it. Inevitably, the requirements will have to be updated or redone; otherwise, we will have another activity, **working from old requirement**s, which clearly causes wasted effort.

Consider **"fixing" bugs**. The quotes around the fixing are intentional: although most developers believe they spend a considerable amount of time fixing bugs, in reality they spend more time looking for the bugs. This is not just semantics. Imagine a developer who writes a bug but is immediately told about it. She can probably fix the problem fairly quickly. Now, consider the situation where the same error occurs, but the developer is not told about the problem for 2 weeks. It will take her considerably longer, even if nothing else changed. It is even worse if it falls on someone else to fix it. Where did this new work come from? It comes from the delay between creating and detecting. I call this "induced work" because it is caused by delay: Just like moving magnets induce electricity in wires, delays induce work in development.

A similar phenomenon happens with **"integration" errors**. Virtually every "integration" error I have seen in my years of development has been caused either by communication errors between two groups or one group not doing what they said they would do. While such an error is detected at integration, it doesn't make it an integration error.

Note how all of these "wasteful"

items, including **building unneeded features** is exacerbated by delays in information or in people being available. **Overbuilding frameworks** is perhaps the only item in the right-hand column that is not caused by delays.

Let's look at the work in Table 1 again. How much time do you spend on the left hand side and how much on the right hand side? Seriously, stop and think about it a minute. In my classes, when I ask people this question, people tell me they spend between 30% and 70% of their time on the left hand side, doing useful work. This means that between 70% and 30% of the time is spent on the right hand side, doing work that is essentially useless![1]

Now, we can speed things up either by doing the work on the left hand side faster or by figuring out how not to do the work on the right hand side at all. While you might be able to do the work on the left hand side faster, with the exception of automated testing, it is not likely you will realize significant improvements doing so. On the other hand, getting better (faster) feedback and having the right people work on the right tasks at the right time can virtually eliminate most of the work on the right hand side.

Shift your focus from productivity to working on the right things at the right time with the right folks. Better productivity and lower costs will come as great by-products. How to do this in software development is exactly what Kanban focuses on.[2]

[1] Overbuilding frameworks is typically caused by lack of knowledge of how to do proper emergent design. The interested reader is referred to Scott Bain's Jolt Award winning book, *Emergent Design: The Evolutionary Nature of Professional Software Development.*

[2] The term MMF (minimal marketable feature) was coined in *Software by Numbers: Low-Risk, High-Return Development* by Mark Denne and Jane Cleland-Huang. It is essentially the smallest piece of value that can be delivered that is worth the transaction cost borne by both the development company and the customer.

WORKING ON THE RIGHT THINGS AT THE RIGHT TIME

So how to shorten the delays involved in our work? In large organizations I would assert that we must include the work done prior to the development teams. We can think of product development flow to have four stages. These are shown in Figure 2.

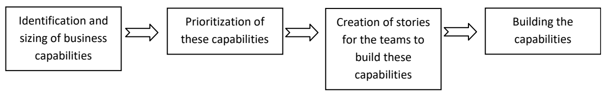

Identification and sizing of business capabilities
Prioritization of these capabilities
Creation of stories for the teams to build these capabilities
Building the capabilities

Figure 2. The stages of product development flow.

Identification and sizing of business capabilities. This is the first step. Business stakeholders (typically business executives or product managers) identify the business capabilities needed. They size these by refining them to their minimum marketable features (MMFs), sometimes called minimum viable features (MVFs).

Prioritization of these capabilities. For organizations with more than one product line, the business stakeholders and product managers that are associated with those products must decide which of these MMFs are the most important. This is product portfolio management. An organization must look across all of its products in order to determine what to work on next. Some products will return most of their value in a short period of time while others may continue to deliver large returns with each investment. It is important that teams not get locked into their products but rather switch to a product with a higher return if it makes more business sense.

Creation of stories for the teams to build these capabilities. Once the MMFs have been prioritized, they need to be broken down further before the teams pull them to actually build the capabilities. It should be apparent that for large organizations this flow will be somewhat complex. Each business capability will likely be split out across several teams which must build different components that have both technical and business dependencies. A technical dependency between components means that one component requires another one to work. A business dependency means that although the components may work without each other, their value won't be realized until all of them work. This is not unlike when you wait for your bags after a flight. If you have three bags, getting the first two doesn't give you any value. It is only when that third bag arrives that you can leave.

A SHORT LESSON IN PULL

To understand the flow in Figure 2, you need to understand a key Lean management concept: pull. Planning work is very difficult in high-variability situations – which software development certainly is. The idea of pull is that rather than planning the timing of our work, we have each step pull work from the output of the previous step. For example, in Figure 2, the development team (depicted as 'building the capabilities') would pull from the work area preceding it that creates the stories. The people doing the work here need just enough work to ensure that when a team is ready to start a project there is enough work there for them to do so. Perhaps a little more for safety's sake since sometimes projects are longer or shorter than expected. In other words, the development teams should be working as fast as they can, working to their capacity, and pulling work whenever ready. The upstream folks know to create another project to be ready when the developers pull one off the ready queue.[3]

By managing with pull, the organization focuses on lowering the queues between the different steps. As the queues get smaller, the work flows faster since the biggest delays in most software development is not so much the work itself as much as the time between the steps. To see this, consider how much time people are waiting for answers or approvals.

[3] This coordination is not usually done in real-time but rather by checking on a regular basis (e.g., weekly) to see if new work needs to be prepared. The regularity of this checking is called the cadence of the step.

THE NEED FOR MANAGEMENT

It should be apparent that trying to coordinate this sequence of events with its accompanying business dependencies and technical dependencies requires a larger perspective than a team view. Not one of micro-management from the top. But one of recognizing global interactions are different than local ones. How do we reconcile these different perspectives – the one from the top that is attempting to make a cohesive flow and the one from the bottom where the actual work takes place? Lean science provides a basis for aligning both global and local efforts because both should be looking to shorten cycle time in whatever they do.

Current management places a lot of emphasis on ensuring that people are working productively. Lean management would have management shift from trying to motivate people to providing them with the proper work environment in which to get their work done. "Work environment," means more than simply the physical environment they work in. It includes the way they integrate with others doing development. Management's role shifts to a larger view: helping to create an organizational structure in which the people doing the work can best function. Instead of managing by telling people what to do, managers now assist teams in getting their work done by providing the proper environment for them to do so and coaching as required.

In large organizations, managers can best assist teams by helping coordinate their work so that they are not over-burdened. Overloading teams is one of the primary causes of serious delay. The Lean manager's role is both to focus on how work flows and in coaching teams to attend to avoid delays. Management provides the high view of the organization needed while providing assistance to work done at the team level.

In Figure 2, management's role could be considered to be getting proper flow of work from the left to the right. This would entail having the right number of things being worked on at each step of the process. Imagine how having the right number of items hitting the team for them to pull from will help their productivity. A secondary effect will be that product owners will be more available to the teams as well. A common challenge in software development organizations is the lack of product knowledge which slows the teams down. It is ironic that the very people who can help them (e.g., product owners) are spending time creating more stories than are necessary. By limiting how much work can be upstream of the team pulling their work, the organization is also simultaneously making these people needed by the development teams more available to them.

LEAN LEARNING

Development systems are very complex. Cause and effect are very difficult to determine. Many events are unpredictable. And since people are involved, events can even be irrational at times. This means we have to learn quickly and challenge our assumptions. Lean's underlying systems thinking approach enables a three step approach for learning.[4] The first is to see where you are. Then, look to see where you want to be. Finally, take steps to get there. This learning approach specifies to take very small steps. If your steps are too large, many things will happen from when you start until you see results. This will make it difficult to see the results of your actions. Kanban methods follow this by using explicit policies (e.g., limitations on work in progress at each step of your process). You can readily see the results of these actions if you have your work reflected in a value stream map or Kanban board.[5]

PUTTING IT TOGETHER

Becoming Lean is truly a journey and not a destination. It involves creating visibility where you are so you can see what you need to do to get to where you want to go. You strive to improve your work by removing delays between the steps of the work. One method of doing this is setting work in progress (WIP) limits so you do not exceed the capacity of your organization at any step. By eliminating delays, you lower induced work which raises true productivity and quality, while lowering cost. Because we are taking a scientific process here, all the roles of the organization can see if our actions are helping or hurting. Overall cycle time is our measure of efficiency of the organization. Local changes can be made with confidence of overall improvement if they lower our overall time. The goal is continuous improvement by improving how we work to get better products for our customers. We learn in small steps so we can understand the results of our actions. Visibility is not just limited to how our work is flowing but includes the rules we use for making decisions. This enables managers to assist development teams since they can work to have the proper organizational structure they need as well as coach the teams when they need it.

Lean Software Development is a combination of Lean science, Lean management and Lean learning. It provides an overall approach that helps different roles in the organization to see how progress is being made, both in the products being built and in the way they are being built By creating visibility and a common language they help create a better, more productive team.

[4] I highly recommend Mike Roth's *Toyota Kata: Managing People for Improvement, Adaptiveness and Superior Results*. While based on Toyota (and therefore manufacturing) the learning process equally applies to knowledge work.

[5] Scrum boards can accomplish this but typically don't as most Scrum teams do not explicitly state how they move work that takes place within a Sprint. Rather they limit their explicit policies to be how stories get on (Sprint ready) and off (done) the board.

COPYRIGHT © NET OBJECTIVES, INC.

LEARN TO DRIVE DEVELOPMENT FROM THE DELIVERY OF BUSINESS VALUE

What really matters to any organization? The delivery of value to customers. Most development organizations, both large and small, are not organized to optimize the delivery of value. By focusing the system within which your people are working and by aligning your people by giving them clear visibility into the value they are creating, any development organization can deliver far more value, lower friction, and do it with fewer acts of self-destructive heroism on the part of the teams.

THE NET OBJECTIVES TRANSFORMATION MODEL

Our approach is to start where you are and then set out a roadmap to get you to where you want to be, with concrete actionable steps to make immediate progress at a rate your people and organization can absorb. We do this by guiding executive leadership, middle management, and the teams at the working surface. The coordination of all three is required to make change that will stick.

OUR EXPERTS

Net Objectives' consultants are actually a team. Some are well known thought leaders. Most of them are authors. All of them are contributors to our approach.

Al Shalloway

Alan Chedalawada

Guy Beaver

Scott Bain

Max Guernsey

Luniel de Beer

SELECTED COURSES

Executive Leadership and Management
- Lean-Agile Executive Briefing
- Preparing Leadership for a Lean-Agile/SAFe Transformation

Product Manager & Product Owner
- Lean-Agile Product Roadmaps
- PM/PO Essentials

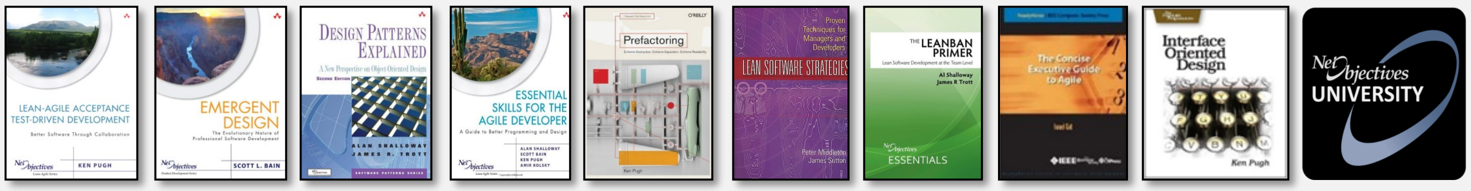
Lean-Agile at the Team
- Acceptance Test-Driven Development
- Implementing Team Agility
- Team Agility Coaching Certification
- Lean-Agile Story Writing with Tests

Technical Agility
- Advanced Software Design
- Design Patterns Lab
- Effective Object-Oriented Analysis and Design
- Emergent Design
- Sustainable Test-Driven Development

DevOps
- DevOps for Leaders and Managers
- DevOps Roadmap Overview

SAFe®-Related
- Implementing SAFe with SPC4 Certification
- Leading SAFe® 4.0
- Using ATDD/BDD in the Agile Release Train (workshop)
- Architecting in a SAFe Environment
- Implement the Built-in Quality of SAFe
- Taking Agile at Scale to the Next Level

OUR BOOKS AND RESOURCES


LEAN-AGILE ACCEPTANCE TEST-DRIVEN DEVELOPMENT
KEN PUGH
EMERGENT DESIGN
SCOTT L. BAIN
DESIGN PATTERNS EXPLAINED
ALAN SHALLOWAY
JAMES R. TROTT
ESSENTIAL SKILLS FOR THE AGILE DEVELOPER
Prefactoring
LEAN SOFTWARE STRATEGIES
THE LEANBAN PRIMER
The Concise Executive Guide to Agile
Interface Oriented Design
Ken Pugh
Net Objectives UNIVERSITY

CONTACT US
info@netobjectives.com
1.888.LEAN-244 (1.888.532.6244)

LEARN MORE
www.NetObjectives.com
portal.NetObjectives.com

Copyright © Net Objectives, Inc.