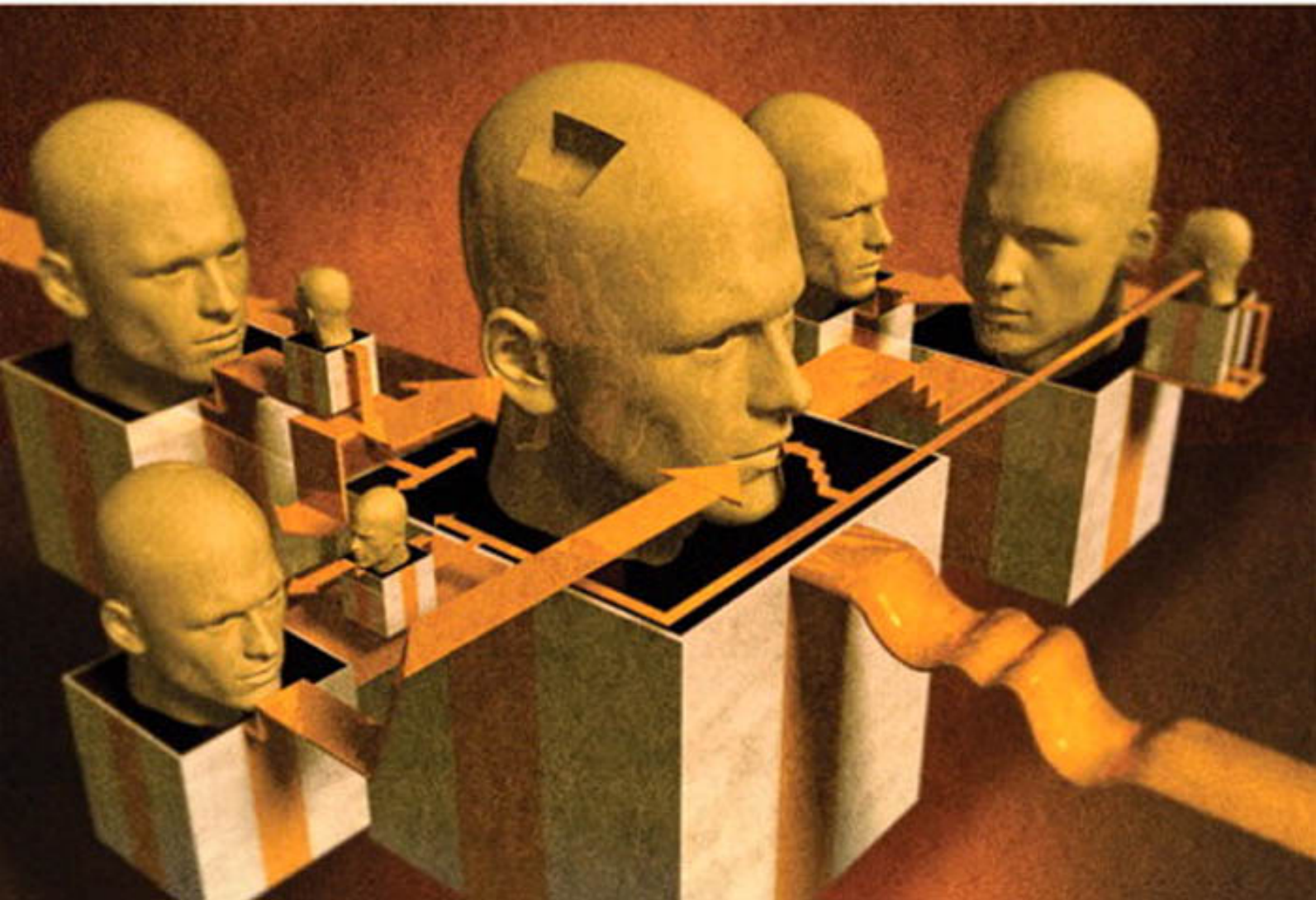
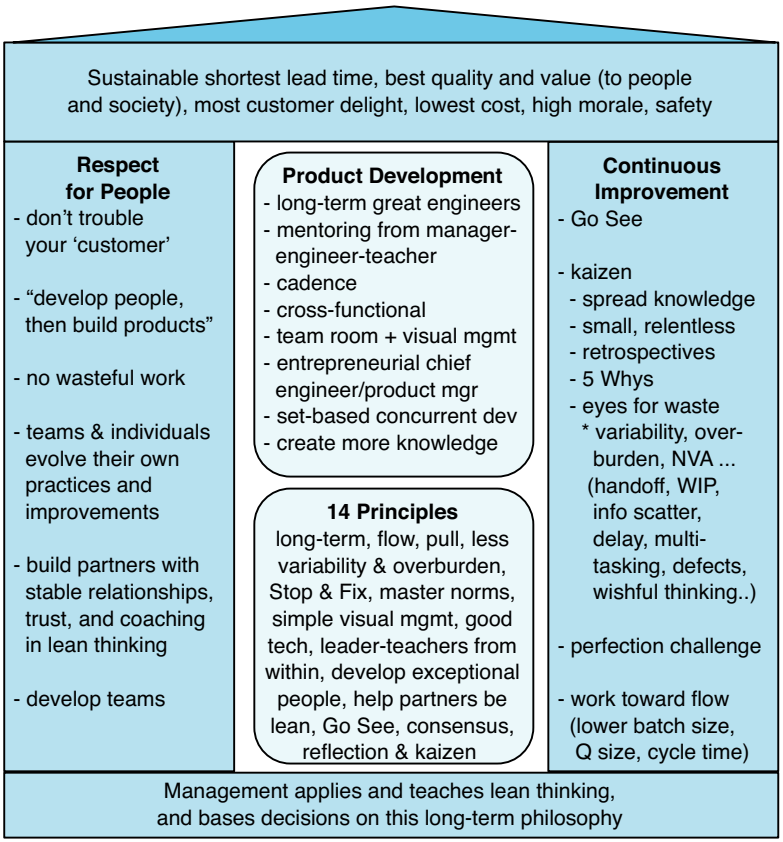
Scaling Lean & Agile Development

Thinking and Organizational Tools for Large-Scale Scrum

Craig Larman
Bas Vodde



THE LEAN THINKING HOUSE



Lotus 1-2-3?—Some readers may not have even heard of Lotus 1-2-3, but it once *owned* the spreadsheet market. Borland and Microsoft came out with competing products with better graphics. Lotus was slow to respond (long cycle times), and three years later the competitors held 52% of the market share, about $500 million USD in sales. Lotus 1-2-3 RIP. [Meyer93]

leveling p. 65

Toyota people understand statistical variation and the implications of queueing theory; this is reflected in the lean *leveling* principle to reduce variability and more generally in the lean focus on smaller batches and cycle times to move toward *flow.* As will be evident, Scrum supports the management implications of queueing theory.

Before diving directly into the subject, note that lean is sometimes described as focusing on *smaller batch (work package) size*, *shorter queues*, and *faster cycle time*. Delivering value quickly. Lean is much more than this—the pillars are *respect for people* and *continuous improvement* resting on a foundation of *manager-teachers in lean thinking*. Queue management is a mere tool far removed from the essence of lean thinking. That said, faster cycle time is part of the "global goal" in lean: *Sustainable shortest lead time, best quality, most customer delight, lowest cost, high morale, safety.* So, on to cycle time…

TRY…COMPETE ON SHORTER CYCLE TIMES

A lean product development organization is focused on *value throughput in the shortest possible sustainable cycle times*, focused on the *baton* rather than runners. Toyota people, the progenitors of lean thinking, are masters of faster and faster (shorter) cycle times without overburdening people.

What are some process cycles or cycle times in product development?

- "concept to cash" for one release
- "concept to done" for one feature
- potentially shippable time—how frequently *could* you ship?
- integration time (to integrate and fully test the product)
- compile time (of all the software)
- "ready to pilot" to delivery time
- deployment time for testing (into embedded hardware)
- analysis and design times

Try…Use several high-level cycle-time KPIs

Key performance indicators (KPIs) in lean are *not* focused on the utilization of workers doing these processes. Rather, *lean KPIs focus more on throughput cycle times*—the baton rather than the runners.

That said, a caution: Measurement usually generates dysfunction or 'gaming' of the system by sub-optimizing to appear to achieve a good score [Austin96]. This is especially true on 'lower' process cycles. Higher-level cycle times such as potentially shippable cycle time and "order to cash" or "order to delivery" (the quintessential cycle times) are most relevant.

What would it mean if you could deliver in *half or a quarter of the time* at a *sustainable pace* without overburdening people? And on the other hand, what is the cost of delay?

Consider the benefits of delivering fast in terms of life cycle profits, opportunities gained, response to competition, and innovation. For most companies—not all—it would be an *extraordinary* advantage.

Half the time is not half the cost—When people hear "half the time" they may think, "Twice as many products, features, or releases—twice the old efficiency." But there could be more **transaction cost**, the overhead for each cycle. Shipping more frequently might increase testing or deployment costs—or not, as will be seen.

> **Economic model includes cycle time**—How to consider the trade-off of shorter cycles versus transaction costs? Use an economic model of your product that *includes cycle time factors* [SR98]. Suppose you could ship six months sooner. What are the estimated total life cycle profit impact and the increased testing costs (transaction cost)? If you could gain $20 million at a 40 percent increase in testing costs ($1.3 million), it is money well spent. The flip side of this is the

cost of delay. One product study showed a 33 percent loss of total profit due to a six-month delay [Reinertsen83]. Unfortunately, many product groups we work with do not seriously analyze cycle time factors in their life cycle profit economic model.

Half the time is not twice the cost—Before you put away your spreadsheet on the transaction cost analysis, hold on. There is a subtle connection between cycle time, transaction cost, and efficiency that will soon be explored—a secret behind the impressive efficiency of Toyota and other lean thinking enterprises…

Queue management—There are plenty of strategies to reduce cycle time; both lean and agile practices offer a cornucopia of skillful means. One tool is the subject of this chapter—queue management.

Queue Management to Reduce Cycle Time

"*Queues only exist in manufacturing, so queueing theory and queue management don't apply to product development*." This is a common misconception. As mentioned, queueing theory did not arise in manufacturing but in operations research to improve throughput in telecom systems with high variability. Furthermore, many development groups—especially those adopting lean or agile practices—have adopted queue management based on queueing theory insight for both *product development* and *portfolio management*. One study from MIT and Stanford researchers concluded:

> *Business units that embraced this approach [queue management for portfolio and product management] reduced their average development times by 30% to 50%. [AMNS96]*

Queues in Product Development and Portfolio Management

Example queues in development and portfolio management?

- products or projects in a portfolio
- new features for one product
- detailed requirements specifications waiting for design
- design documents waiting to be coded
- code waiting to be tested
- the code of a single developer waiting to be integrated with other developers
- large components waiting to be integrated
- large components and systems waiting to be tested

In traditional sequential development there are many queues of partially done work, known as work-in-progress or **WIP queues**; for example, specification documents waiting to be programmed and code waiting to be tested.

In addition to *WIP queues*, there are **constrained-resource** or **shared-resource queues**, such as a backlog of requests to use an expensive testing lab or piece of testing equipment.

Queues Are a Problem

First, if there are no queues—and no multitasking that artificially makes it appear a queue has been removed—then the system will move toward flow, the lean principle and perfection challenge that value is delivered without delay. Every queue creates a delay that inhibits flows. More specifically, why are queues a problem?

WIP queues—WIP queues in product development are seldom seen as queues for several reasons; perhaps chief among these is that they tend to be *invisible*—bits on a computer disk. But they are there—and more importantly they create problems. Why?[1]

- ❑ WIP queues (as most queues) *increase average cycle time* and reduce value delivery, and thus may lower lifetime profit.

1. See also the *Recommended Readings* for a cogent analysis of queues in product development and what to do about them.

- In lean thinking, WIP queues are identified as *waste*—and hence to be removed or reduced—because:
  - WIP queues have the aforementioned impact on cycle time.
  - WIP queues are *inventory* (of specifications, code, documentation, …) with an investment of time and money for which there has been no return on investment.
  - As with all inventory, WIP queues hide—and allow replication of—defects because the pile of inventory has not been consumed or tested by a downstream process to reveal hidden problems; for example, a pile of un-integrated code.
  - We saw a traditional product group that spent about one year working on a "deal breaker" feature. Then product management decided to remove it because it threatened the overall release and the market had changed. Replanning took many weeks. In general, WIP queues *affect the cost and ability to respond to change* (deletions and additions) because (1) time and money were spent on unfinished deleted work that will not be realized, or (2) the WIP of the deleted item may be tangled up with other features, or (3) a feature to add can experience a delayed start due to current high WIP levels.

benefits of reducing cycle and batch p. 112

As will be explored, there is a subtle but potentially powerful systems-improvement side effect that can occur through the process of eliminating WIP queues.

Shared resource queues—In contrast to WIP queues, these are more often seen as queues—and seen as a problem. They clearly and painfully slow people down, delay feedback, and stretch out cycle times. *"We need to test our stuff on that printer. When will it be free?"*

Try…Eradicate queues by changing the system

The bottom line is that (usually) *queues are a problem*. Given that, you may jump to the conclusion that the first line of defense against this problem is to *reduce the batch and queue size*, because these are classic queue-management strategies. Yet, there is a *Gordian Knot* solution that should be considered first…

The remainder of this chapter will indeed explore reducing cycle time through batch- and queue-size management. But that entire management strategy should be *Plan B*. Rather, start with *Plan A*:

> Plan A in queue management is to *completely eradicate the queue, forever, by changing the system—of development, tools*, …

Think outside the current box and shorten cycle times by changing the *system* so that queues no longer exist—by removing bottlenecks and other forces that create the queues. These constraints and the queues they spawn may be created or eradicated by the very nature of a development system and its tools.

Suppose the existing system is based on sequential or serial development with single-specialist workers or groups. There will be WIP queues: The analyst group hands off specification work packages to the programming group that hands off code work packages to the testing group. The *inside-the-box* response to improving cycle time with queue management is to reduce batch size, reduce variability, and limit the WIP queue sizes between these groups.

But there is a deeper alternative that will more dramatically improve cycle time: Discard that system and the bottlenecks and WIP queues it spawns. If you adopt cross-functional feature teams that do complete features (analysis, programming, and testing) without handing off work to other groups, and that apply automated acceptance test-driven development (TDD), the above WIP queues *vanish* by moving from serial to parallel development.

feature teams p. 149

Avoid…Fake queue reduction by increased multitasking or utilization rates

Suppose you are busy working on item A, and items B, C, D, and E are in your queue. Fake queue reduction is to work on *all* these items at more or less the same time—a high level of multitasking and utilization. Multitasking is one of the lean wastes because as will be soon seen, queueing theory shows that this would *increase* average cycle time, not reduce it. Bad idea.