Leading Lean Software Development – Results Are Not the Point

Mary and Tom Poppendieck; Addison-Wesley, 2010, 278 pages
ISBN 0-321-62070-4

Book report
Jim McDonough, January 31, 2010

Review[a]

This is the Poppendiecks' third book on the subject of lean software development, completing a trilogy beginning with their 2003 publication Lean Software Development – An Agile Toolkit and continuing with their 2007 work Implementing Lean Software Development – From Concept To Cash. It supplements the preceding books delving more deeply into the management aspects of software development. Continuing a trend established with their first book, this one also is packed with helpful suggestions, insightful analysis and practical advice.

After listing some corporations notably succeeding with lean practices, the authors state this purpose:

> "The purpose of this book is to explore how we might adapt the way these great companies frame the role of leadership to organizations involved in developing software-intensive systems." (p. xvi)

The book is organized into six chapters, each presenting four of what the authors call "frames", perspectives into different aspects of managing software development. Each chapter is arranged using a common format:

- A short snapshot of the topic to be covered.
- An overview of the topic, illustrated using an historical example.
- The four frames, each offering a different perspective on the topic.
- A portrait of a leadership role associated with software development.
- A trailer section titled "Your shot", in which probing questions are presented for contemplation by the reader.

As to the use of the frames metaphor, the authors explain:

> "According to cognitive scientists, we all interpret our surroundings through frames – mental constructs that shape our perspective of the world. Frames are sets of beliefs about what elements to pay attention to and how these elements interact with each other. Frames place significant limits on our perspective; we can see only what our frames tell us is meaningful, and we usually ignore what lies outside the boundaries." (p. xvii)

Though it covers many facets of the software industry, the book concentrates primarily on the management of software development. Indeed, this book, it seems, is to management of software development what W. Edwards Deming's The New Economics[b] is to management practices in general, the Poppendieck's use of frames facilitating what Deming called a lens into the four parts of his System of Profound Knowledge (Appreciation for a system; Knowledge of variation; Theory of knowledge; Psychology),[c] with the authors devoting particular attention to the fourth part - Psychology.

---

[a] Footnote references using alphabetic characters refer to passages cited by the author of this review. Those using numbers denote actual footnote references accompanying the corresponding text cited in the reviewed book.

[b] Deming, "The New Economics for Industry, Government, Education", 2nd edition, 1994.

[c] Ibid., pp. 92-93.

1. Systems Thinking

W. Edwards Deming taught that a system is a network of interdependent components working together to accomplish a common aim, and that a system must be managed.[d] The chapter Systems Thinking applies the system concept to the process of software development. Southwest Airlines is cited as an example, illustrating how the company, abandoning the conventional wisdom of how an airline should be run, has been able to sustain profitability while maintaining high customer satisfaction. This is achieved by having a corporate culture emphasizing the optimization of the entire system and not each of the system's interdependent components. The authors note Southwest Airlines routinely flies with airplane load factors of 90%, compared to the 100% goal of its competitors.

Frame 1, **Customer Focus**, presents a perspective for considering the value customers derive from a system, exploring the question "Who are your customers?", explaining:

> "A customer is anyone who *pays for, uses, supports, or derives value from* the systems you create. Thus you probably have a lot of customers. But wait; there's more. If you are working on a subsystem of a larger system, you have customers of your subsystem and customer of the overall system. So arriving at clarity about who your customers are can be challenging." (p. 6)

This section gives consideration to the question "What is the nature of customer demand?", subdividing demand into value demand and failure demand:

> "Customer demand comes in two forms. First there is demand for products and services that provide value. Second, when a product or service doesn't meet the customer's expectations, there is demand for failure remediation – that is, demand to fix something that appears to be broken, inadequate, or difficult to use, configure, or modify." (p. 10)

Frame 2, **System Capability**, reveals the concepts associated with system predictability, offering recommendations on how to determine the capability of the system:

> "If you want to know how well your organization is performing, don't just look at a few data points; look at a sequence of data over time. All systems exhibit variation, and looking at an occasional data point does not tell you much about how much variation there is in your system. If fact, random data points give a distorted view of your current capability." (p. 13)

adding the caution:

> "You should *not* set targets. Your current system is what it is; targets are not going to change its capability. You *measure* system capability; you do not prescribe it. As Deming once said, 'If you have a stable system, then there is no use to specify a goal. You will get whatever the system will deliver. A goal beyond the capability of the system will not be reached. If you have not a stable system, then there is no point in setting a goal. There is no way to know what the system will produce; it has no capability'".[15] (pp. 15-16)

Frame 3, **End-toEnd Flow**, renders a wide view of the entire system and begins:

> "Far too often, providing customer value is thought of as a series of separate input-process-output steps, rather than an integrated flow of work through an organization. But if no one takes an end-to-end view of a customer problem or a product as it moves through a work system, customer problems fall into the cracks between departments, hard-earned knowledge evaporates at handovers, and the things that customers will really value get lost along the way. Developing an

---

[d] Ibid., pp. 95-96.

[15] Deming, *Out of the Crisis*, 2000, p. 76.

understanding of the end-to-end flow of work through a work system is fundamental to systems thinking." (p. 19)

Frame 4, **Policy-Driven Waste**, exposes the sources of waste caused by an organization's policies and procedures, describing:

> "Waste is anything that depletes resource of *time, effort, space*, or *money* without adding customer value." (p. 24)

The authors identify the five biggest causes of policy-driven waste: (p. 26)

- Complexity
- Economies of scale
- Separating decision-making from work
- Wishful thinking
- Technical debt

Each cause is discussed in further detail.

The portrait presented at the end of this chapter is a sketch of the Product Champion, a role in software development compared with a chief engineer in manufacturing.

2. Technical Excellence

Exploring various bandwagon fads popular in their day as well as concepts and techniques that have withstood the test of time, the chapter Technical Excellence begins as a retrospective on the history of software engineering from its inception to the present. Of particular interest is a section titled the Life Cycle Concept, where it is revealed that, while it began in the early 1970s and is so prevalent today in the form of the classic waterfall process, one of its first criticisms was made public in the 1982 article Life Cycle Concept Considered Harmful, written by Dan McCracken and Michael Jackson, two early pioneers of data processing. Mary and Tom Poppendieck subsequently observe:

> "As we look at the history of software development, we see a consistent goal: Avoid the high cost of discovering defects later in the development process by discovering them as early in the development process as possible. However, we see a big difference of opinion on how to go about achieving this goal. On the one hand we have people like Dijkstra and Mills, who framed establishing correctness as an ongoing activity of system development. On the other hand, we have the life cycle concept, which framed establishing correctness as a project management activity. We recall what Daniel McCracken and Michael Jackson said in 1982: 'Any form of life cycle is a project management structure imposed on system development'". (pp. 61-62)

Frame 5, **Essential Complexity**, encompasses various approaches to managing the complexity inherent in software. In a section titled Conway's Law,

> "'Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations,' Mel Conway wrote in 1968.[36]" (p. 67)

the authors propose a reason why some companies find adapting to change so difficult:

> "One of the reasons why high-dependency architectures seem so intractable is that they usually reflect the organization's structure, and changing the organizational structure is often not considered an option. But it should be. We regularly find companies struggling to implement agile practices with functional teams, that is, teams organized along technology lines. For

[36] Conway, "How Do Committees Invent?," 1968.

example, there might be a database team, a mainframe team, a Web server team, even a testing team. Invariably these technology (or functional) teams find that they are dependent on the work of several other teams, so delivering even a small feature set requires a large amount of communication and coordination. The complexity of the communication adds time and increases the likelihood of error, and the added overhead cost makes small incremental releases impractical.[37]" (p. 68)

offering this warning:

> "It is difficult for agile development to flourish in a highly interdependent environment, because teams cannot independently make and meet commitments. Agile development works best with cross-functional teams in which a single team or small group of teams has the skill and authority necessary to deliver useful feature sets to customers independent of other teams. This means that whenever possible, teams should be formed along the lines of features or services. Of course, this is most useful if those features or services are relatively small and have clean interfaces. Conway's Law is difficult to escape." (p. 69)

Frame 6, **Quality by Construction**, focuses on how various testing methodologies offer significant benefits toward improving quality, emphasizing the discipline necessary for maximizing the benefits to be gained by following agile methods:

> "Part of the reason why we delay testing is because testing has historically been labor-intensive and difficult to perform on an incomplete system. With a lean frame of mind, we can no longer accept this historical mental model; we must change it. In fact, changing the historical mind set regarding testing is probably the most important characteristic we have seen in successful agile development initiatives. You can put all the agile project management practices you want in place, but if you ignore the disciplines of test-driven development and continuous integration, your chances of success are greatly diminished. These so-called engineering disciplines *are not optional* and should not be left to chance." (p. 71)

Frame 7, **Evolutionary Development**, encapsulates a cycle of discovery, short iterations designed to provide quick feedback, and extols the merits of set-based design and development:

> "Evolutionary development is the preferred approach for reducing risk, because it confronts risk rather than avoiding it. Evolutionary development attempts to discover the systems that are best able to handle risk either by rapidly adapting the current system to handle risks as they present themselves, or by fielding multiple systems and selecting the best alternative." (p. 84)

Frame 8, **Deep Expertise**, reflects upon the characteristics of expertise and why it is so important that it be cultivated on software development projects.

The portrait presented at the end of this chapter is a profile of the Competency Leader, the role played by a seasoned domain expert capable of mentoring those who themselves aspire to become domain experts.

3. Reliable Delivery

Describing how it was predominately the team approach, decoupling of dependencies and concentration on flow that proved so successful in enabling it to be completed on time, the chapter Reliable Delivery reflects upon the methods and techniques used in constructing the Empire State Building.

---

[37] See Larman and Vodde, *Scaling Lean and Agile Development*, 2009, Chapter 3, Feature Teams, for an in-depth discussion of this topic.

> "Without decoupling, the task of managing the construction of the Empire State Building would have been impossible. Today we use computers to help us with complex scheduling problems, but perhaps that is a mistake. We are tempted to create a tightly interlocking schedule, forgetting that the slightest deviation will necessarily cascade throughout the entire interlocked system. Not only is a decoupled schedule much simpler, but the impact of variation is isolated to its own workflow." (p. 106)

Frame 9, **Proven Experience**, covers the effect that established expertise has on system design, constraints and design loop-backs. Also exposed are three areas where common scheduling practices can fail – dependencies, utilization and critical path. On making a commitment for what can be delivered and when, the construction of the Empire State Building holds a lesson:

> "The commitment to deliver on time and on budget was not made based on the details; the commitment was made based on the *ability to shape* the details. This is very important. *The schedule was not laid out based on the details of the building design; the building was designed based on the constraints of the schedule*." (p. 108)

Frame 10, **Level Workflow**, envelops an analysis of batch size – how big is a software release – and presents two methods for managing workflow: iterations and kanban. The authors offer the following advice to managers for determining the capacity of the software development organization:

> "In order for work to flow, it is very important to limit the amount of work to capacity; do not try to force more work through your development organization than it has the capacity to deliver. Trying to force a system to work beyond its capacity invariably causes turbulence and slows things down. It is rather difficult to limit work to capacity, however, if you do not know what your capacity it is. Whether you use iterations or kanban, it is important to generate a useful measurement of your capacity, so you know how much work your system is capable of handling." (p. 128)

Frame 11, **Pull Scheduling**, explores scheduling work for small, medium and large systems. The lesson of the proven experience by the builders of the Empire State Building is reiterated here:

> "The builders did not lay out a detailed schedule of the building based on its design; the building was designed based on the constraints of the schedule. In other words, the builders did not work backward from the details of the building to arrive at its schedule; we would call this *push* scheduling. Instead, they started with the constraints of the system and developed the schedule to accommodate those constraints; we call this *pull* scheduling." (p. 129)

Frame 12, **Adaptive Control**, depicts the benefits of frequent feedback:

> "The most robust systems are designed to continually monitor their own performance, compare it to the ideal, and adapt their parameters based on this feedback, to move performance closer to the ideal. The essence of adaptive control is to put good feedback mechanisms in place and use the feedback effectively. (p. 143)

The portrait presented at the end of this chapter offers a different perspective on the Product Champion, this time featuring actual product champion Chris Hughes, the visionary usability guru behind Facebook.

4. Relentless Improvement

Deming observed that about 94% of most trouble and possibilities for improvement belong to the system, and proclaimed: "No amount of care or skill in workmanship can overcome fundamental faults of the system."[e] The chapter Relentless Improvement focuses on continuous process improvement and begins

[e] Deming, "The New Economics for Industry, Government, Education", 2nd edition, 1994, pp. 33-34.

by relating the experiences of dedicated professionals who worked to improve the system by which medical care is administered at a Pennsylvania hospital. In a compelling reminder of how easy it is for us to accommodate the problems of a system rather than resolve them, the section titled No Work-arounds tells of a lone nurse who decided to make a change:

> "Gloria[5] was a nurse in West Penn Allegheny Hospital who went to a talk by Alcoa CEO Paul O'Neill about Alcoa's goal of zero injuries. O'Neill said that the way to approach the goal was to *see problems, solve problems and share the learning*. Gloria always thought of herself as a problem solver, but that day she realized that she had been solving the same problems over and over, every day, for the last 20 years. She decided to stop working around problems and get rid of them once and for all." (p. 156)

The authors caution against blindly copying what works for others:

> "… adopting 'best practices' from other organizations means you are adopting their solutions to *their* problems. It is far more effective to learn how to devise your own solutions to *your* problems." (p. 159)

Frame 13, **Visualize Perfection**, draws attention to subconscious perceptual limitations often constricting the potential for process improvement, and highlights the differences between low- and high-velocity companies:

> "… high-velocity companies *do not try to predict* how to deal with complexity; they *focus on learning* about the complexity. This is a very important distinction. Consider a development schedule. A low-velocity company would think of a schedule as a plan and measure variance against that plan, engaging in the static process of *predicting* what it will take to meet the goal. High-velocity organizations are not in the business of making predictions; instead, they would regard a schedule as a *hypothesis* of what it might take to meet the goal. When there is a variance from schedule, the hypothesis would be disproved; *it would be obvious that the schedule isn't perfect*. A high-velocity organization *welcomes variance as a learning opportunity*, a way to discover more about the complexities of its work. High-velocity organizations do not focus on getting the work done; they focus on *learning how to get work done*. There is a huge difference" (pp. 161-162)

Frame 14, **Establish a Baseline**, orients the reader for beginning to contemplate how work is currently done in the organization. Handovers are identified as places where work moving through the process falls through the cracks, those spaces between the work specialties where attention by management is usually not part of the process. Reinforcing how some recent management trends eviscerate knowledge workers of their ability to improve their own work processes is this eloquent narrative from the section titled Process Standards:

> "Any discussion of work design inevitably brings up the controversial issue of process standards. In an article titled 'The Coming Commoditization of Process,'[14] Thomas Davenport writes that the purpose of process standards is to enable clear communication and clear handoffs across the business; as a result, it is possible to create comparative performance measures and outsource work more easily. Davenport claims that in particular, CMMI has had an enormous impact on improving the quality of software as well as facilitating the outsourcing of software development. Framing process standards in this manner carries the implicit assumption that once processes are sufficiently commoditized (de-skilled), anyone should be able to do any job – so people become fungible resources. This framing of process standards tends to be accompanied by the belief that it is important to roll out the same processes to all parts of a company.

---

[5] From Spear, *Chasing the Rabbit*, 2009, and Spear, "Fixing Health Care from the Inside, Today," 2005, pp. 341-46.

[14] Davenport, "The Coming Commoditization of Processes," 2005.

> "We believe that this is an unfortunate distortion of the purpose of effective process standards. We have seen little evidence that pushing a centrally devised development process across all locations in a company is particularly effective, or that de-skilling development work leads to better systems. On the contrary, we observe that some of the best systems around – the Internet, the Google Search Engine, the Amazon Cloud – are the result of empowering local teams to act with courage and creativity to meet challenges and to independently devise and improve their own processes.
>
> "Process standards had their origins in the scientific management movement of the early 1900s and rested on the theory that process experts and supervisors were the best qualified to design and control the work of individual workers in the pursuit of the most efficient output. A half century later, when Japanese products started outperforming their competitors in both price and quality, process standards were revitalized. It was observed that top Japanese companies followed detailed process standards, so efforts to copy this practice spread. Assessment approaches such as ISO 9000 and CMM/CMMI focused on assuring that companies had formal standards in place that were explicit and rigorously followed. But this framing of process standards led many companies to miss the real reason for standards: The purpose of process standards is to act as a baseline for continuous improvement. It is the overhead incurred to enable auditability that induces the waste, not the standards themselves." (pp. 167-168)

Frame 15, **Expose Problems**, illustrates the tendency to accentuate the positive and disregard the negative, noting that many organizations, to their detriment, have established a corporate culture which discourages workers from being messengers with bad news. As Deming asserted: "Fear invites wrong figures. Bearers of bad news fare badly. To keep is job, anyone may present to his boss only good news."[f]

A better approach is to have an atmosphere where problems are raised freely and discussed openly in an effort to improve the process that allows for problems to occur:

> "The purpose of problem solving is not so much to fix a problem as it is to really understand more about the complex system of work flowing through your organization and learn how to make things work better.
>
> "The most important thing in a continuous improvement environment is to expose problems, look for them, learn how to see them and welcome them. Do not ignore problems or think of them as background noise. Problems are simply the work process telling you that you don't understand it well enough, so pay attention and learn." (pp. 171-172)

Frame 16, **Learn to Improve**, provides a glimpse of a variety of methods by which to begin the process of understanding how to learn, including the A3 report, the Five Whys, fishbone diagrams and the PDCA cycle (Plan; Do; Check; Act).[g] Mary shares this in a sidebar on Suggestion Systems:

> "Forget suggestion systems where people abandon their annoyances to a box and become increasingly cynical when no one notices how inane their work process is. Instead, provide mentors for everyone who is annoyed, someone who will guide them through the steps it takes to get rid of the annoyance once and for all. You will find that complaining clock punchers turn into passionate problem solvers when they are the ones responsible for fixing the things that make their job difficult." (p. 176)

---

[f] Deming, "The New Economics for Industry, Government, Education", 2nd edition, 1994, p. 94.

[g] The PDCA cycle also is known as the PDSA cycle, where Study replaces Check. Furthermore, though commonly known as the Deming cycle, it originally was conceived by Walter A. Shewhart, Deming's mentor, and Deming himself had referred to this as the Shewhart Cycle (See Deming, "Out of the Crisis", 2000, p. 88, and "The New Economics for Industry, Government, Education", 2nd edition, 1994, p. 132).

The portrait presented at the end of this chapter showcases the Manager as Mentor, a role central to facilitating process improvement.

5. Great People

In explaining Psychology, one of the four pillars of his System of Profound Knowledge, Deming observes: "People are different from one another. A manager of people must be aware of these differences, and use them for optimization of everybody's abilities and inclinations."[h] The chapter Great People expounds upon this concept with a fascinating analysis of how cultural influences assimilated since childhood by people of diverse nations affects behavior in the workplace. After listing the 12 principles behind the Agile Manifesto, the authors speculate on the relative comfort of embracing each principle based upon one's cultural background, observing:

> "The way we discuss lean software development is similar to and supportive of the principles of agile software development, and it certainly has an American bias. But lean thinking has a Japanese heritage, and it is this heritage that makes lean concepts counterintuitive to Westerners. Consider the principles:
>
> 1. Eliminate Waste
> 2. Build Quality In
> 3. Create Knowledge
> 4. Defer Commitment
> 5. Deliver Fast
> 6. Respect People
> 7. Optimize the Whole
>
> "Probably the most challenging of these from a Western perspective are principle 4, 'Defer Commitment,' and principle 7, 'Optimize the Whole.' To countries with a long-term orientation, these are natural ideas. Delivering fast seems like a good idea to Western ears, until it becomes apparent that speed requires abandoning deeply held beliefs such as maximizing utilization. …
>
> "Lean principles – whether in manufacturing, logistics, or development – seem counterintuitive and internally inconsistent in cultures with a short-term orientation. They tend to create dissonance in places where a focus on short-term results feels like the only rational approach. This is probably why lean initiatives have been so difficult to implement in a sustainable manner in Western cultures." (pp. 193-194)

Frame 17, **Knowledge Workers**, offers a viewpoint to the productivity of those who build software, explaining the title of the frame with a nod to its originator:

> "Peter Drucker, sometimes called 'the man who invented management,'[12] … coined the term *knowledge worker* in 1959 to emphasize the fact that knowledge workers were becoming an increasingly large portion of the labor force. By 'knowledge workers' Drucker meant people who create value through mental rather than manual effort." (p. 196)

After listing the six major factors that, according to Drucker, determine the productivity of knowledge workers, the Poppendieck's deduce:

> "… it should be blindingly obvious why we struggle when trying to apply management practices developed for manual labor to knowledge work. Most of these practices – concepts such as high utilization and low variation, identical processes and fungible resources – are the *exact opposite*

---

[h] Deming, "The New Economics for Industry, Government, Education", 2nd edition, 1994, p. 108.

[12] Byrne, "The Man Who Invented Management: Why Peter Drucker's Ideas Still Matter," 2005.

of what is needed to increase the productivity of knowledge workers, as Drucker points out." (p. 197)

The authors challenge management practices focusing on results with this retort:

> "In knowledge work, success comes entirely from people and the system within which they work. Results are not the point. Developing the people and the system so that together they are capable of achieving successful results is the point.
>
> "'Fine,' you say, 'but I have software to deliver, budgets to stay within, schedules to meet.' Indeed you do. But here's the thing: Either your system is capable of delivering those results or it is not. Recall that in Frame 2: System Capability we recommended that you start by creating a time series chart of your current process capability. It will tell you what results your system is capable of. Trying to force results beyond what your development system is capable of might achieve results in the short term, but it will invariably make things worse over the long term.
>
> "You are not going to improve your system by stressing it to achieve aggressive targets. Systems don't work that way. You may get a local or temporary optimization, but you will not get a system capable of delivering high-quality software reliably, repeatedly on time." (pp. 198-199)

Frame 18, **The Norm of Reciprocity**, examines attitudes exhibited by workers toward work policies established by management.

Frame 19, **Mutual Respect**, is a panorama covering cross-cultural teams, diversity and self-organizing teams, an acknowledgement that software development lends itself to collaborative efforts spanning the world.

Frame 20, **Pride of Workmanship**, casts a psychological perspective into the potential of the workforce when barriers are removed and workers can take pride in their work. The phrase 'Pride of Workmanship' refers to the 12th of Deming's famous 14 Points,[i] and the authors begin with a Deming tale:

> "W. Edwards Deming used an interesting technique to create respect between workers and their supervisors. He used to start out a consulting engagement by meeting with workers without supervisors present, but with a tape recorder. Then he would ask the workers to describe their frustrations at not being able to do their jobs the way they would like to do them. Usually the managers were shocked when they heard the tapes. They heard their workers say they knew their jobs depended on productivity and quality, understood what it took to do their jobs well, and yet were allowed neither to do a good job nor to fix the problems that kept them from doing work they could be proud of.[29] So began the education of supervisors about how to respect their subordinates by removing barriers to pride of workmanship." (p. 209)

The portrait presented at the end of this chapter illustrates Front-Line Leaders, those who make critical decisions as events unfold, and presents a convincing argument for why leadership skills are necessary not just at the top, but also at all levels of an organization.

6. Aligned Leaders

In his book The New Economics, Deming describes the role leadership played in the late 1930s in transforming the survey methods used by the US Bureau of Census.[j] The chapter Aligned Leaders similarly describes the leadership exemplified in maneuvering IBM through the transformation of

---

[i] Deming, "Out of the Crisis", 2000, pp. 23-24 and pp. 77-85.

[29] Walton, *The Deming Management Method*, 1986, p. 81.

[j] Deming, "The New Economics for Industry, Government, Education", 2nd edition, 1994, the chapter titled Leadership, pp. 116-120.

establishing a software development process founded on agile principles, an effort as well as a product known within the company as Agile@IBM. As a case study, the IBM example offers lessons in both the potential benefits as well as the inherent difficulties in changing a business process.

> "'This was the most successful development initiative rollout I have ever seen at IBM,' Paul Gibson commented, and he has seen a lot of rollouts. 'Agile@IBM has been embraced, while most new processes seem to encounter resistance,' someone else commented. We think we know why. First, although Agile@IBM is based on the same approach and measurements across the company, it is being pulled by development teams rather than pushed at them. Second, the teams are in direct contact with their real stakeholders,[3] and able to do what stakeholders really want and nothing more, and are given regular feedback from stakeholders who really care about what they are doing. That makes it fun to come to work every day. And finally, the disciplined approach to developing high-quality code that is essential for successful iterative development means that developers are proud of the software they produce." (p. 221)

A transformation to agile practices is not without its challenges:

> "A key problem was the static business process, a governance process that required detailed approval of content prior to development. The transformation team often heard from students in its classes that this process was simply incompatible with agile, and yet it was not going to disappear. So the transformation team worked to change the business process in two ways. First, the features committed to during the approval process would now be high-level descriptions, with details to be determined during development. And second, each approval list had to have both mandatory and "run-at"[7] features; that is, there had to be space for stakeholder feedback to change the plan. Other corporate groups also had to change to accommodate agile development. The legal department had to figure out how to support the early access program. Testing, HR, globalization, translation, and other business processes had to be adapted. The transformation team worked to help these groups and others understand how their policies might inhibit agile development." (p. 224)

Emerging from the IBM experience is this profound lesson, the significance of which easily can be overlooked by many managers recently discovering agile methods:

> "The other big lesson was that line managers really need to understand agile in order to guide their teams to success. Trust is not enough if teams and their leaders do not understand how to go about successful agile development. Starting agile development without the necessary technical discipline and tools in place is dangerous, and no manner of goodwill and trust can save a bad implementation of agile." (p. 224)

Frame 21, **From Theory to Practice**, spotlights the distinction between good theory and good theory that is implemented well.

> "One lesson from the case study stands out above all of the rest: Early and frequent feedback from stakeholders (customers) directly to those who are developing the system – without intermediaries – fosters pervasive alignment by focusing everyone on customer outcomes." (p. 228)

Frame 22, **Governance**, captures the essence underlying traditional management methodologies used in many large organizations.

---

[3] *Stakeholders* in this section means "the people who will ultimately engage with and benefit from the product: (Kessler and Sweitzer, Outside-in Software Development: A Practical Approach to Building Successful Stakeholder-Based Products, 2008, p. 2). We use the term *customers* to refer [to] the same people throughout the rest of this book.

[7] Features the team will take a "run at" but will not guarantee to finish.

> "One of the biggest obstacles that had to be dealt with when implementing agile development at IBM was the governance process that required detailed approval of product content prior to development. This is likely to be one of the biggest obstacles to any large-scale implementation of lean development, because the governance process for system development in most large organizations is based on creating detailed project plans and measuring performance against those plans. This is true whether the systems are developed under contract between firms or development is done completely within a vertically integrated organization. The governance process in many companies boils down to the equivalent of negotiating a contract for development and measuring performance against that contract. As long as this governance process remains in place without modification – whether inside of or between firms – the full potential of lean development processes will not be realized." (pp. 230-231)

The section titled Beyond Budgeting presents an interesting perspective on the problems associated with the budgeting process.

> "What's wrong with budgets? In the book *Implementing Beyond Budgeting*, Bjarte Bogsnes counts the ways:[10]" (p. 231)

On prediction:

> "Budget targets, as well as project plans, are usually laid out in great detail, and the variance between plan and actual is measured at the same detailed level. The amazing thing is the assumption that it is possible to guess, at such a detailed level, long in advance, exactly what the right thing to do will be." (p. 232)

On trust:

> "Most performance measurements that are based on detailed budgets and plans telegraph a deep lack of trust and need for control. … The control systems are in place to keep the few untrustworthy people in the company in line, yet they signal a lack of trust of everyone, telling them they aren't expected to act in a trustworthy manner." (p. 231)

Worker behavior becomes a consequence of management policies, and when a conflict arises, compliance with rigid governance criteria will trump professional judgment:

> "… people are not evaluated on whether or not they made the best, most effective, most value-generating decision at the time; they are evaluated on whether or not they followed – in detail – the long-outdated guesses from the past." (p. 232)

Frame 23, **Alignment**, magnifies the introspection into whether all members of a leadership team share the same perception of cause and effect.

> "If you are considering a lean initiative, your leadership team would do well to discuss the members' individual beliefs about cause and effect: *If we do **this**, we can expect **that** result*. Before you start down the path of change, make sure that the members of the leadership team are in substantial agreement on cause and effect." (p. 237)

Frame 24, **Sustainability**, offers a telescopic view, prompting consideration for maintaining constancy of purpose into the future, avoiding the pitfalls which often cause successful initiatives to decay over time.

> "Leadership alignment is two-dimensional: First, the members of the current leadership team must align their perspectives; and second, the leadership perspective has to be maintained over time, and not just a short time – over decades of time. It's pretty obvious that this is not easy. No

[10] Bogsnes, *Implementing Beyond Budgeting: Unlocking the Performance Potential*, 2009. These points are summarized in pp. 8-52 of this book.

matter what name or initials a change initiative has – TQM, CMMI, Lean, Six Sigma – the tendency of these initiatives to plateau after harvesting the low-hanging fruits is well documented, and the rate of relapse is high." (p. 242)

The portrait presented at the end of this chapter depicts Leaders at All Levels, reinforcing the idea that leadership should be abundant and evident throughout an organization.

Summary

This is a superb addendum to their other books. Like the first two books, it is relatively short and easy to read. The bibliography lists more than 8 full pages of other publications referenced throughout the text, ranging from Adam Smith's An Inquiry into the Nature and Causes of the Wealth of Nations (1776) to Bjarte Bogsnes' Implementing Beyond Budgeting: Unlocking the Performance Potential (2009), with nearly two thirds of these having been published since 2000.

Notable passages

"The primary demand on your organization should be value demand. The demand can be in the form of requests for work that will add value from a customer's perspective, or it can be in the form of unmet needs that customers don't known they have, but that you discover and satisfy through your products and services. Value demand, when satisfied, usually generates revenue for your organization." (p. 12)

"Customers don't usually see change approval systems as a value; they see the *changes they need* as a value, but asking for permission to make the change is annoying. If customers see your change request approval process as a nuisance, that process is handling failure demand." (p.21)

"… queues were not the responsibility of any department manager; their real purpose was to buffer departments from the variation of neighboring departments. Failure to see the impact of queues is often caused by focusing exclusively on department-level performance or by trying to achieve high utilization of scarce talent. These policies leave most companies oblivious to the tremendous drag local optimization has on time-to-market and end-to-end flow and hence on cost, revenue, and yes, even utilization." (p. 26)

"Visualization leads to evaluation, a series of quick experiments that incrementally improve the product. 'It doesn't matter how clever you are, your first idea about something is never right,' says Tim Brown, CEO of IDEO. 'So the great value of prototyping – and prototyping quickly and inexpensively – is that you learn about the idea and make it better.'"[41] (p. 40)

"Develop a clear vision of how the product will meet market needs, how the architecture will support that vision, and how development will proceed. We call this a *product concept*. A product concept is not a detailed plan; it is a framework for proceeding with development." (p. 41)

"Virtually all software development methodologies share the same objective: Lower the cost of correcting mistakes by discovering them as soon as they are made and correcting them immediately. However, as we demonstrated earlier, various methodologies differ dramatically in the way they frame this goal. The sequential life cycle approach is to create a complete, detailed design so as to find defects by inspection before any coding happens – because fixing design defects before coding is supposedly less expensive than fixing design defects later on. This theory has a fundamental problem: It separates design from implementation. A careful reading

[41] Brown, "The Deans of Design," 2006.

of the 1968 NATO conference minutes shows that separating design from implementation was regarded as a bad idea even as the term *software engineering* was coined."[39] (p. 70)

"Executable tests (specifications, actually), whether they are acceptance tests or xUnit tests or other tests, are the building blocks of mistake-proofing the software development process. These tests are put into a test harness, and then every time new code is written, the code is added to the system and the test harness is run. If the tests pass, you know not only that the new code works, but also that everything that used to work is still working. If the tests don't pass, it's clear that the last code you added has a problem or at least has exposed a problem, so it's backed out and the problem is immediately addressed." (p. 74)

"Many companies set aside 30% or more of each release cycle for a series of integration tests and the associated repairs necessary when the tests uncover defects. On the other hand, the best companies budget no more than 10% of a release cycle for final verification, and they do this with increasingly short release cycles. They can do this because the defects have been uncovered earlier in the development process; if they find defects at final verification, they are surprised and work to discover ways to keep those defects from escaping to the final verification step in the future." (p. 77)

"The biggest cause of failure in software-intensive systems is not technical failure; it's building the wrong thing. We know this. Study after study confirms it. And yet we continue to put intermediaries between the development team and its customers. We might call these intermediaries systems analysts, or product owners, but all too often they represent a handover of information. Secondhand information about the problem to be solved handicaps the people making the detailed decisions about how the system will behave. A critical role for product managers and product owners is to build bridges between customers and development team members, so they can talk to each other." (p. 86)

"The bottom line of evolutionary development is this: You cannot afford to develop systems of a size and timeline that stretch beyond your environment change time. If you are using lengthy planning processes to increase predictability, you can confidently predict this: If the system development time is greater than the environment change time, the system will fail." (p. 88)

"When you read something written in your native language, you can quickly tell what's being said, but when you look at writing in a language you don't understand, you have no clue what it's all about. Programmers write in a peculiar language, and they write for people who understand that language; those who can't read the language aren't part of the audience. Sometimes we see demands for documentation from people who can't read the code, but we know that anyone who seriously expects to change the code is going to have to read it. We believe that for most purposes, it is much more important for code to speak for itself than for documentation to translate its meaning." (p. 89)

"If writing software were a translation process, you could hire a couple of great designers, have them design a great system, give the design to a team of translators, and expect excellent results. But attempts to do this have a dismal track record. By now we should have learned that in software development, we cannot separate design from implementation." (p. 90)

"The only people who can reliably commit to deliver are those people who have developed a reliable capability to deliver. You don't want to sum up the estimates of dozens of people and assume that their estimates will add up to a reliable whole. They probably won't. You want people who have the demonstrated capability to reliably deliver whole systems to lead the effort; and you want them to figure out how to design the details of the system so that the constraints of the system are met in the best possible way. You do not work backward from the details of a

[39] Naur and Randell, Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, 1968.

system to its schedule; you start with the high-level schedule and design the details of the system to meet that schedule. *This strategy of designing the effort to fit the constraints, rather than computing the constraints from the design, is absolutely the most effective way to achieve reliable delivery.*" (pp. 108-109)

"Sequential development methods, derived from contracting practices, encourage over-the-wall handovers, even though it has been demonstrated many times that concurrent engineering – engaging experts from all steps of the workflow in the initial design – is one of the best ways to eliminate design loop-backs." (p. 111)

"If you want to go fast and meet deadlines, you have to figure out a way to simplify and decouple schedules. Unfortunately, many common scheduling practices decompose system development in a way that will almost certainly *increase*, rather than decrease, coupling." (p. 112)

"Most schedule coupling that we see in software development does not come from the architectural dependencies; it is the direct result of trying to achieve a high utilization of people. It seems that many companies don't ever want to see idle workers, so they use scheduling systems to enable them to fully use worker's available time. This creates tight coupling of multiple schedules and invariably leads to cascading delays. The inevitable delays decrease utilization to a far lower point than what can be achieved with a decoupled scheduling system. As we have demonstrated in previous books[12], queuing theory makes it clear that attempting to maximize utilization is a self-defeating process. Optimal utilization can be achieved only by concentrating on flow." (p. 113)

"If your schedule is mission-critical, every element of the so-called critical path should have at least one and preferably two alternate approaches, insofar as that is possible. This is called *set-based design*, a practice we have covered in previous books.[13] The idea of set-based design is to develop multiple alternatives for every critical design decision and make the decision as late as possible – when you have the most information. At decision time, you must be sure that at least one of the alternatives will work. You might develop a good approach that will work for sure, a better approach that has bit more risk and an aggressive approach that pushes the limits but would be the best approach if it works out. Developing multiple options may seem more expensive than developing a single approach, but for critical decisions it is almost always cheaper and gives better results." (p. 113)

"When a group of teams delivers reliably based on the teams' velocity, you begin to get a good feel for the capacity of the organization. This is a relatively rough estimate, and it pretty much falls apart if you reorganize teams frequently. Velocity varies from one team to another, and it changes when the team makeup changes or when the team moves to a different class of problems." (p. 128)

"The constraints exist. Deadlines are usually very important to a business. The amount of money that can be invested to achieve a benefit has valid economical limits. On the other hand, in the overwhelming majority of cases, scope in a software system is not only expendable; it is usually overblown and selected by people who do not accept accountability for, or have the knowledge needed for, fitting the work within the constraints. Let's be honest; customers do not need scope. They need to have business goals accomplished within some constraints of time and cost. So it is imperative that the real constraints of time and cost be used to limit scope, and this imperative holds true in almost every situation we have encountered in software development." (p. 138)

---

[12] See Poppendieck, Lean Software Development: An Agile Toolkit, 2003, Chapter 4; and Poppendieck, Implementing Lean Software Development: From Concept to Cash, 2006, Chapter 5.

[13] See Poppendieck, Lean Software Development: An Agile Toolkit, 2003, pp. 38-45; and Poppendieck, Implementing Lean Software Development: From Concept to Cash, 2006, pp. 160-64.

"In the book *What Is Quality Control? The Japanese Way*, Kaoru Ishikawa writes: 'Standards and regulations are imperfect. They must be reviewed and revised constantly.' 'If newly established standards and regulations are not revised in six months, it is proof that no one is seriously using them.' 'Detailed standards and regulations are useless if they are established by headquarters staff and engineer-specialists who do not know or do not try to know the workplace and who ignore the wishes of the people who have to use them.'"[16] (p. 169)

"Many compensation systems, especially in America, rank individuals against each other for the purpose of determining pay raises. Under such a system, the best way to earn more money is to know as much as possible while being compared against others who know as little as possible. This creates the best incentive we have ever seen for knowledge hoarding rather than knowledge sharing." (p. 182)

"'Managers in high-velocity organizations make sure that a regular part of their work is both the delivery of products and services and also the continual improvement of the processes by which those products and services are delivered. They teach people how to make continual improvement part of their jobs and provide them with enough time and resources to do so,' says Steven Spear. 'High velocity managers are not in place to command, control, berate, intimidate, or evaluate through a contrived set of metrics, but to ensure that their organizations become ever more self-diagnosing and self-improving, skilled at detecting problems, solving them and multiplying the effect by making the solutions available throughout the organization.'"[31] (p. 183)

"Conceivably, executives could insist that the number of support calls be reduced and set a target – for example, cut support calls in half. However, setting a target does not do much to change the number of support calls; the only way to reduce the support calls is to change the system: Change one or several of the processes used in developing the software that generates a lot of support calls." (p. 229)

Useful links

- http://agilemanifesto.org/principles.html
- http://www2.toyota.co.jp/en/vision/traditions/may_jun_06.html
- http://collaborativeleadership.com/about.html
- http://www.sae.org/manufacturing/lean/column/leanjul01.htm
- http://www.poppendieck.com

---

16 Ishikawa, *What Is Quality Control? The Japanese Way*, 1985, pp. 62 and 56.

31 Spear, *Chasing the Rabbit*, 2009, p. 26.