11 Understanding Agile at a Deeper Level

AN AGILE PROJECT MANAGER needs to understand agile at a deeper level in order to apply it to different situations effectively. The key to that is to develop a systems thinking approach to understand agile principles and practices at a deeper level. In order to develop that kind of systems thinking approach, it is valuable to understand the roots of agile and how agile thinking evolved. The roots of agile go fairly deep, but there are two major sources that had the most impact on its development:

- Total quality management (TQM) was probably the strongest factor in influencing the agile approach to quality.
- Lean manufacturing was probably the biggest factor in influencing agile process thinking.

Each of those influences will be discussed in this chapter.

SYSTEMS THINKING

BusinessDictionary.com defines *systems thinking* as follows:

> Practice of thinking that takes a holistic view of complex events or phenomenon, seemingly caused by myriad of isolated, independent, and usually unpredictable factors or forces. Systems Thinking views all events and phenomenon as 'wholes' interacting according to systems principles in a few basic patterns called systems archetypes. These patterns underlie vastly different events and phenomenon such as diminishing returns from efforts, spread of contagious diseases, and fulfillment in personal relationships.

> Systems Thinking stands in contrast to the analytic or mechanistic thinking that all phenomenon can be understood by reducing them to their ultimate elements. It recognizes that systems ('organized wholes') ranging from SOAP bubbles to galaxies, and ant colonies to nations, can be better understood only when their wholeness (identity and structural integrity) is maintained, thus permitting the study of the properties of the wholes instead of the properties of their components.[1]

In the context of agile, *systems thinking* means understanding the principles behind the methodology rather than just focusing on the mechanics of how the methodology works and understanding how those principles interact with the overall project and business environment that they are part of. Why is systems thinking important? It allows you to see things in an entirely different perspective:

- You see the whole rather than the pieces and understand their relationship. In an agile implementation you see the business as a large ecosystem and see the development process as only one component of that ecosystem and you begin to better understand how the two are interrelated to each other.
- Within an agile development process, you begin to better understand how all the components of that process work together to make the overall process more effective and instead of following the process rigidly and mechanically, you see it as a much more dynamic process where each component of the process may need to be adjusted to fit the situation.

Binary thinking is the antithesis of systems thinking. Instead of seeing the real complexity that is inherent in many situations, people who engage in binary thinking are sometimes looking for a simple, cause-effect explanation for something that isn't really very simple at all:

- They tend to see the agile values and principles in black-and-white terms, as absolute statements, rather than relative statements that need to be interpreted in the context of the situation as they were intended to be.
- They see the relationship of agile and more traditional plan-driven approaches as either-or, mutually exclusive choices (Either you're agile or you're not) and they may see these approaches as competitive with each other rather than seeing them as potentially complementary.

That sort of narrow thinking has led to many stereotypes, myths, and misconceptions about what agile is, and also about what traditional project management is. We need to rethink what agile is as well as rethink what traditional project management is to see them in a new light as potentially complementary rather than competitive approaches. Systems thinking is the key to that.

[1] "What Is Systems thinking?" http://www.businessdictionary.com/definition/systems-thinking-ST.html#ixzz2z3A07AVt.

System thinking is closely related to the idea of a learning organization. Business Dictionary.com defines a *learning organization* as follows:

> Organization that acquires knowledge and innovates fast enough to survive and thrive in a rapidly changing environment. Learning organizations (1) create a culture that encourages and supports continuous employee learning, critical thinking, and risk taking with new ideas, (2) allow mistakes, and value employee contributions, (3) learn from experience and experiment, and (4) disseminate the new knowledge throughout the organization for incorporation into day-to-day activities.[2]

Systems thinking provides a mechanism to understand the dynamics behind how an organization works at a deeper level. The culture of a *learning organization* creates an environment where that information is used for ongoing, continuous improvement. Adopting a *systems thinking* approach and becoming a learning organization are two of the most important aspects of achieving enterprise-level agility.

INFLUENCE OF TOTAL QUALITY MANAGEMENT (TQM)

In order to understand agile at a deeper level, it is useful to understand the roots of agile and how those principles have evolved in different environments. Many of the agile principles related to quality have their roots in the philosophy of total quality management (TQM). The TQM philosophy originated from the ideas of W. Edwards Deming and others. Dr. Deming was an American statistician who was credited with the rise of Japan as a manufacturing nation. His principles transformed the Japanese automotive industry into developing very-high-quality products that gained significant market share against American automotive manufacturers in the 1970s and 1980s (see Figure 11.1).

Dr. Deming's original 14 points can be summarized into five major areas that have a significant impact on an agile project management approach. The following are a summary of some of Dr. Deming's original 14 points that the TQM philosophy is based on that form the roots of today's agile approach for software development. According to Deming, "The 14 points all have one aim, to make it possible for people to work with joy."[3]

1. Cease dependence on inspection
2. Emphasis on the human aspect of quality
3. The need for cross-functional collaboration
4. Importance of leadership
5. Ongoing continuous improvement

[2] "What Is a Learning Organization?" http://www.businessdictionary.com/definition/learning-organization.html#ixzz2z3C2Bv5s.
[3] Alexander Laufer, *Mastering the Leadership Role in project management: Practices that Deliver Remarkable Results* (Upper Saddle River, NJ: Pearson Education, 2012).

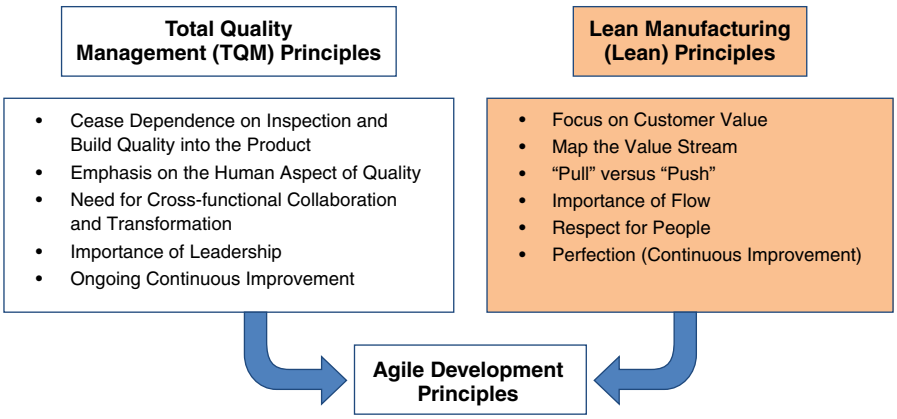

FIGURE 11.1 The roots of agile practices

Each of these points and how it impacts an agile project management approach will be discussed in the following sections.

Cease dependence on inspection

Deming's Principle:	Point #3—"Cease dependence on inspection by building quality into the product in the first place."

Manufacturing Problems Prior to TQM	**Manufacturing TQM Approach**
Prior to TQM, manufacturers relied heavily on quality control inspectors to detect problems at the end of the assembly line. The problems with that approach are: ■ If problems are not detected until the end of the assembly line, it can be expensive to go back and rework or scrap the product at that point. It also can have a significant impact on cycle time to wait for a product to be reworked before it can be shipped.	A better approach is to go upstream in the process, find the sources of error that contribute to the defects, and eliminate those sources of error at the source. That approach results in: ■ Better quality because the quality is designed into the product from the beginning, and defects are prevented before they happen

(*Continued*)

Manufacturing Problems Prior to TQM	Manufacturing TQM Approach
■ Any inspection approach like this relies heavily on sampling; and, with a limited number of inspectors, it's very time-consuming to do a very large sample size. For that reason, it's difficult to fully test every possible product or situation that could result in a defect. As a result, some defects are bound to slip through and result in very poor quality, as seen by the customer. ■ Quality is seen as the responsibility of the inspectors, who are perceived as the *enforcers*. As a result, the people who are producing the products may not feel fully responsibility for the quality of the products they produce. ■ It is expensive to employ enough quality control inspectors to perform this function completely, and even with lots of inspectors, it still might not be very effective.	■ Reduced costs because it relies less on inspection to find problems ■ Greater pride of ownership by the workers who are primarily responsible for producing the product

Implications for Agile Development:

An agile development approach recognizes and incorporates this principle. Instead of performing QA testing sequentially with development and relying heavily on a typical quality assurance approach to detect problems after development has been completed and sending the product back for rework to fix defects (bugs), it is far more effective to make quality an integral part of the development process, do it more concurrently with development, and prevent the bugs (defects) from happening at the source. This approach also results in reduced costs because it relies less on inspection (QA) to find problems after the product development is complete.

Agile makes producing quality products the responsibility of the team developing the product, which typically includes QA testers. It's not someone else's responsibility (like a separate QA department) to ensure that the product is free of defects. This increases pride of ownership of everyone on the team and results in much higher quality products.

Emphasis on the human aspect of quality

Deming's Principles:

- Point #6: "Institute training on the job."
- Point #8: "Drive out fear."
- Point #12: "Eliminate barriers to pride of workmanship."
- Point #13: "Institute education and self-improvement."

Manufacturing Problems Prior to TQM

In the early days of manufacturing, *sweatshops,* where people worked under oppressive conditions, were quite common. It's obvious that people who are overworked and who aren't respected and recognized for the importance of the work they do are probably not going to be highly motivated and are likely to produce a much lower quality product. In particular,

- People who are fatigued from working very long hours in a less than ideal working environment are prone to make errors.
- People with narrowly defined and repetitive jobs, who only have a limited responsibility for a small portion of the overall product (such as putting the bolts on a wheel on an automobile), may have difficulty feeling ownership for and taking pride in the overall product they are producing. That may affect the quality of the product they produce.
- People who are primarily motivated by fear of the consequences of *not* performing a task well are generally not going to work as effectively as people who are *positively* motivated to produce a high quality product because they take pride in their work.
- No one wants to be a *cog in a wheel*. People want to see a higher purpose and value in the work they do, and they want to be respected and recognized for their work.

Manufacturing TQM Approach

Manufacturing companies learned a long time ago to pay attention to the human aspects of producing quality products:

- Improving working conditions and automating menial, repetitive tasks as much as possible, rather than relying on people to perform those tasks, uses the skills of people more effectively and removes a major source of errors and defects.
- Engaging people at all levels of production through *quality circles* and other mechanisms so that they feel responsibility for the quality of the overall products they produce, rather than just their own particular role in producing the product, will ultimately lead to more pride of ownership and higher quality products.
- Providing training and education to all employees, building their skills to perform the process at a higher level, and empowering them to recognize and suggest opportunities for improvement in the process are essential for ongoing process improvement.

(*Continued*)

Implications for Agile Development:

An agile software development approach recognizes this by:

- Making respect for people a very important value.
- Fully engaging everyone on an agile development team as an equal contributor to the success of the product.
- Putting a high level of emphasis on the training and skill of individuals to exercise good judgment in how the process is executed rather than relying on highly prescriptive, predefined processes to tell people what to do.
- Substituting positive motivation and leadership for traditional command-and-control management. The Scrum Master on a team is a facilitator, not a directive manager, and should empower everyone on the team to be fully engaged in the process.
- Eliminating sweatshops and Death March projects where people are forced to work excessive amounts of overtime to meet arbitrary schedule commitments and instead working at a *sustainable pace*.

The need for cross-functional collaboration and transformation

Deming's Principles:

- Point #2: "Adopt the new philosophy."
- Point #9: "Break down barriers between departments."
- Point # 14: "The transformation is everyone's job."

Manufacturing Problems Prior to TQM	Manufacturing TQM Approach
Prior to TQM, responsibilities were typically split across different organizations (engineering, production, quality, etc.). As a result: ■ Each organization was typically focused on their individual objectives, and the responsibility for the overall effectiveness and quality of the process was fragmented.	Moving to a TQM approach required a major shift in thinking using *systems thinking* to see the whole business from a much broader process perspective (rather than a hierarchical organizational perspective) and to develop a well-integrated cross-functional approach across all of the organization. Although this can

(*continued*)

(*Continued*)

Manufacturing Problems Prior to TQM	Manufacturing TQM Approach
■ It can be very difficult to break down these barriers to develop a much more integrated and unified approach.	be difficult to achieve, the results are significant: ■ It eliminates conflicting goals among organizations and develops an integrated cross-functional approach that leads to much higher levels of productivity and efficiency. ■ It enables everyone in the company to see an overall vision of how the products and projects they're producing provide value to leverage the company's business success and how their individual role contributes to that goal.

Implications for Agile Development:

An agile development approach addresses this by emphasizing self-sufficient and autonomous cross-functional teams as the focal point for responsibility and decision making to break down organizational barriers at the project level. Even that doesn't go far enough, in many cases.

A major problem with the implementation of an agile development process is that it is often perceived as just a software development process owned by the development organization. People incorrectly ignore the need for organizational transformation and senior management commitment to make it an integral part of the way the business operates to make it fully successful.

Importance of leadership

Deming's Principle: Point #7: "Institute leadership"—Supervision should help people and machines do a better job. Supervision of management is in need of overhaul as well as supervision of production workers.

Manufacturing Problems Prior to TQM	Manufacturing TQM Approach
Traditional command-and-control styles of management may not be very effective, even in a manufacturing environment. There are a couple of significant problems in that approach: ■ It doesn't fully empower and engage employees to take an active role in processes that are more self-managed. ■ It can be very demotivating to employees to work in that kind of environment where their skills are not recognized and valued and without more effective leadership.	Truly inspirational leaders help people see the higher-level purpose and vision for their work. As a result, people ■ Are much more highly motivated; ■ Feel more ownership of the products they produce; ■ Work more effectively; and ■ Use their initiative with less need for direct supervision.

Implications for Agile Development:

In a software development environment where creativity and innovation are extremely important, traditional command-and-control management styles may stifle that initiative and creativity.

Ongoing continuous improvement

Deming's Principles:
- Point #1: "Create constancy of purpose toward improvement of products and services with the aim of becoming competitive, staying in business, and providing jobs."
- Point #5: "Improve constantly and forever. Constantly improve quality and productivity in order to constantly decrease costs."

Manufacturing Problems Prior to TQM	Manufacturing TQM Approach
Processes that rely heavily on a reactive approach based on correction of defects for quality control, rather than a more proactive approach based on	With the advent of TQM and Six Sigma in the 1980s and early 1990s, companies learned that they could "push the envelope" much further into developing higher levels

(*continued*)

(*Continued*)

Manufacturing Problems Prior to TQM	Manufacturing TQM Approach
prevention of defects and continuous improvement, will not significantly improve their defect rate because they have not removed the root cause of the defects.	of quality that were never believed possible before that time. It required a totally different and much more systemic approach to eliminate the source of defects and prevent them from happening in the first place, rather than fixing them after they've happened.

Implications for Agile Development:

Continuous improvement is a major focus of all agile methodologies. The need for continuous improvement is done through retrospectives at the end of each sprint or iteration. Because the iterations are relatively short, learning and process improvement can take place rapidly.

INFLUENCE OF LEAN MANUFACTURING

Where TQM provides a foundation of how to integrate quality into the design of products, lean manufacturing principles (see Figure 11.2) complement and go beyond that by developing a stronger focus on *maximizing customer value and providing guidance on how to improve and streamline processes to eliminate wasteful inefficiencies*.

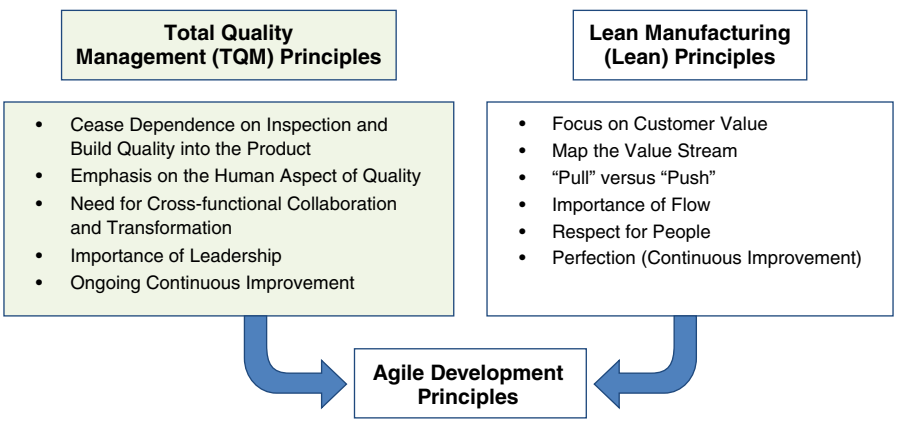

FIGURE 11.2 The roots of agile principles (lean)

Lean manufacturing or lean production, which is often simply known as Lean, is defined as:

> A systematic approach to identifying and eliminating waste through continuous improvement by flowing the product at the demand of the customer.[4]

Lean considers the expenditure of resources for any goal other than the creation of value for the end customer to be wasteful and a target for elimination. Value is defined as any action or process that a customer would be willing to pay for. Agile is based on taking that same thinking from lean manufacturing and applying it to a software development process. It involves looking at a software development process and making critical decisions about whether each activity in the process adds value. There are three kinds of work in any process:

1. *Value-added:* Process steps that produce value the customer is willing to pay for or are essential to directly meeting customer requirements
2. *Non-value-added:* Process steps that are not directly required to produce customer value but are required for other reasons, such as meeting regulatory requirements, company mandates, and legal requirements,
3. *Waste*: Process steps that consume resources but produce no value in the eyes of the customer

Applying these concepts to a software development lifecycle model requires evaluating the various steps in the process and making a judgment of whether they really produce value in the eyes of the customer or not. Dr. David Rico provides a definition of *lean systems engineering* as follows:

> Lean (lēn): Thin, slim, slender, narrow, adequate, or just-enough; without waste
>
> - A **customer-driven** systems engineering process that delivers the maximum amount of **business value**
> - An economical systems engineering way of **planning** and **managing** the development of complex systems
> - A systems engineering process that is **free of excess waste**, capacity, and non-value-adding activities
> - **Just-enough**, just-in-time, and right-sized systems engineering **processes**, **documentation**, and **tools**
> - A systems engineering approach that is **adaptable to change** in customer needs and market conditions[5]

[4]Lean Manufacturing Guide, http://www.leanmanufacturingguide.com/.
[5]David F. Rico, "Lean and Agile Systems Engineering," http://davidfrico.com.

INCOSE[6] has developed a list of systems engineering enablers to support the six key principles of lean:

Lean Principle	Enablers
Value	Focus on delivering customer value: ■ Use a defined process for capturing requirements focused on customer value. ■ Establish the value of the end product or system to the customer (What are the business objectives?). ■ Frequently involve the customer.
Map the Value Stream	Use a well-defined methodology for executing projects: ■ Poor planning is the most notorious reason for wasteful projects. ■ Plan to develop only what needs developing. ■ Plan leading indicators and metrics to manage the project.
Pull	Tailor the process to the risks and complexity of the project to achieve maximum efficiency: ■ The *pull principle* promotes the culture of tailoring tasks and pulling them and their outputs based only on legitimate need and rejecting others as waste. ■ Pull tasks and outputs based on need, and reject others as waste.
Flow	Eliminate bottlenecks that are likely to obstruct or delay progress: ■ In complex programs, opportunities for the progress to stop are overwhelming, and it takes careful preparation, planning, and coordination effort to overcome them. ■ Clarify, derive, prioritize requirements early and often during execution. ■ Frontload the architectural design and implementation. ■ Make progress visible to all. ■ Use the most effective communications and coordination practices and effective tools.
Respect for People	Build an organization based on respect for people: ■ Nurture a learning environment. ■ Treat people as most valued assets.

[6] "International Council on Systems Engineering: Lean Systems Engineering Working Group," http://cse.lmu.edu/about/graduateeducation/systemsengineering/INCOSE.htm.

(*Continued*)

Lean Principle	Enablers
Perfection	Strive for excellence and continuous improvement in the software development processes: ■ Use lessons learned from past projects for future projects. ■ Develop perfect communication, coordination, and collaboration policy across people and processes. ■ Use effective leadership to lead the development effort from start to finish. ■ Drive out waste through design standardization, process standardization, and skill-set standardization. ■ Use continuous improvement methods to draw best energy and creativity from project teams.

Each of these topics is discussed in more detail in the following sections:

Customer value

Many businesses focus heavily on financial results rather than customer value as a primary goal. Without understanding the cause-and-effect relationships that drive those financial results, you're always in reactive mode. When the financial results for the current quarter go down, there's a lot of scrambling around to figure out what went wrong and what caused that to happen and then trying to fix the problem. That's a very reactive approach.

A more proactive approach is to develop an understanding of the factors that drive financial results for your business and focus on those factors. If you focus on customer value and the factors that influence customer value and you do that successfully, the financial results should follow; it's a much more proactive, reliable, and consistent approach for managing a business successfully. The idea is that managing the inputs to a process and managing the process itself is always a better approach than simply trying to manage the outputs of a process.

As an example, I once worked for a company that was known for putting enormous pressure on project teams to meet schedule deadlines. If a project was unsuccessful, there was a good chance that the whole project team might be fired. The problems with that approach should be very apparent—taking a *brute force* approach to try to force the outputs of a process to produce the desired results without an understanding of how the process works or the factors that influence the outputs is not likely to be very effective.

Map the value stream

An important principle of both lean manufacturing and lean software development is to map the value stream. This process basically involves starting from the point that the product or service is delivered

to the customer (either internal customer or external customer) and working backward from that point to map all the process steps that led up to fulfilling that customer value. The next step is to identify and differentiate steps that produce value to the customer from steps in the process that produce no value to the customer and may constitute waste.

An important factor is to identify *waste*. Mary Poppendiek has translated the seven wastes found in a manufacturing process to the equivalent seven wastes found in software development:[7]

The Seven Wastes of Manufacturing	The Seven Wastes of Software Development
Inventory	Partially done work
Extra processing	Extra processes
Overproduction	Extra features
Transportation	Task switching
Waiting	Waiting
Motion	Motion
Defects	Defects

Naturally, in order to manage the sources of waste in a process, you have to have a somewhat defined and documented process that is followed fairly consistently.

Pull

One of the key differences associated with lean is the difference between a *push approach* and a *pull approach*—this difference is also found in most agile approaches. In a traditional manufacturing process approach, production output is forecast, inventory is stocked at various points, and raw materials are then pushed through the process to fulfill that forecast. This type of process has been the predominant approach used in manufacturing for many years to maximize the efficiency of the production equipment used in the process. This approach has three potential deficiencies:

1. The forecasting process requires an intelligent guess at what the customer demand will be well in advance of when it is actually expected to be delivered from production. This is very difficult to do accurately and is fraught with lots of potential problems.
2. The process is very difficult to adjust—if there is a change in customer demand, it can take a considerable amount of time and effort to re-plan the entire process to adjust to that change. There also may be a considerable lag associated with restocking material to support the revised plan.

[7]Tom and Mary Poppendiek, *Lean Software Development—An Agile Toolkit* (Reading, MA: Addison-Wesley, 2003), p. 4.

3. If the forecast is wrong, there are significant potential risks, including:
   - *Winding up with a significant amount of unusable inventory that might have to be scrapped* (In a software project, unusable inventory translates to extra features that no one needs or is likely to use that complicate the product and cause unnecessary maintenance if they are not removed.)
   - *Not having sufficient inventory to fill customer demand if the forecast is wrong* (In a software project, this translates to not having the right features to satisfy customer needs.)
   - *Having to stockpile or store inventory beyond the originally planned duration* (In a software project, this translates into undesirable product management overhead.) "I've often seen this administrative burden lumped onto the project manager who must now wade through bloated scope matrices, backlogs of change requests and unwieldy specifications."[8]

Most traditional plan-driven product development processes such as the waterfall process are based on a similar *push process*. In a software development process, requirements are equivalent to raw materials in a manufacturing process. All the requirements are gathered up front and are pushed through the rest of the development process in a sequential fashion just like a manufacturing assembly line. In a traditional plan-driven product development process, the *push approach* may have the advantage of optimizing the utilization of the resources in a development process if the requirements are relatively certain and known in advance, but that is often not the case. If there is uncertainty in the requirements, it can result in serious potential problems and inefficiencies. Those potential problems and inefficiencies are similar to those associated with a push manufacturing process:

- The product requirements for a new product development effort are essentially a *forecast* of the requirements for a product or application that a customer will need in the future. These requirements may be very uncertain and not very well-defined. Attempting to forecast (or guess at) product requirements well in advance of when the product will actually be deployed and used has even more risk than forecasting production output in a manufacturing process. In addition to the normal risks of forecasting customer demand, the customer may not really know what he/she wants without seeing the product and seeing firsthand how it works. For that reason, the requirements might easily change over that time.
- The process also can be difficult to adjust when changes in customer requirements occur. Typically, many assumptions are made about what the customer requirements are, and elaborate plans, resource assignments, and documentation will be created to support those assumptions.
- There is also a significant risk that the assumptions in the requirements are wrong and don't reflect the real needs of the customer. This may not be discovered until the final product is ready for final acceptance testing.

[8]Erik Gottesman, Personal e-mail comments on book review.

Inaccurate or changing customer requirements may result in a significant amount of lost time and effort to replan around a different set of requirements and assumptions. It might also require substantial rework. If a typical change control system is used, the process for doing that may be very cumbersome and difficult. Both lean and agile approaches avoid these problems by:

- Deferring the resolution of uncertain requirements until a decision is required (when that particular requirement is *pulled* into development for further processing). Avoiding guessing at the requirements up front and waiting until more information is known will typically result in better decisions and avoid many of the problems associated with inaccurate and changing requirements.
- Lean and agile systems openly acknowledge and are built around the assumption that requirements are uncertain and are likely to change as the project progresses. Many traditional processes do not recognize or acknowledge an appropriate level of uncertainty that is actually inherent in the requirements and attempt to superimpose a rigid control model on top of a very uncertain environment.

A *pull* system works by producing only the required amount to meet demand at each stage. In a manufacturing system, this would be characterized by a just-in-time production scheduling system. Many of the ideas for lean manufacturing came from the Toyota Production System and Kanban. If the word *Kanban* were translated literally, *Kan* means visual, and *ban* means card or board. The idea is based on inventory demand cards that are sometimes used in a manufacturing system:

> Picture yourself on a Toyota production line. You put doors on Priuses. You have a stack of 10 or so doors. As you keep bolting them on, your stack of doors gets shorter. When you get down to 5 doors, sitting on top of the 5th door in the stack is a card—a Kanban card—that says build 10 doors. Well it may not say exactly that—but it is a request to build exactly 10 more Prius doors.
>
> You pick the Kanban card up, and run it over to the guy who builds doors. He's been waiting for you. He's been doing other things to keep busy while waiting. The important thing here is that he's NOT been building Prius doors. He takes your Kanban card and begins to build doors.
>
> You go back to your workstation, and just a bit before your stack of doors is gone, the door guy comes back with a stack of 10 doors. You know that Kanban card is slid in between doors 5 & 6. You got the doors just in time.[9]

The process flow works in a similar way for the rest of the plant—when the guy who makes the doors for the Prius runs out of parts that he needs to build the doors, he has a similar Kanban card to

[9]Kanban Development Oversimplified, http://www.agileproductdesign.com/blog/2009/kanban_over_simplified.html.

request more parts from the process that provides those parts to him. The whole process flow is pulled by *actual* customer demand, rather than being pushed by a forecast of someone guessing at what they *think* the customer demand is. Of course, in many situations, this process is computerized, and there are no physical cards used to signal demand.

In an agile software development process, there is a direct analogy between Kanban cards and user story cards. User stories are high-level descriptions of capabilities that the system needs to provide. The following is an example of a user story:

> "As a banking customer, I need to be able to withdraw funds from my account through an ATM machine."

User stories are typically defined early in the project to identify the capabilities that the system must provide, with a sufficient level of detail to do only a rough estimate of the level of effort associated with each. The details of how the user story will be implemented will generally be deferred until it is time to do the design:

- Instead of attempting to define all of the requirements in detail, typically only the high-level requirements are defined upfront to the level of user stories without a lot of detail.
- Instead of treating all requirements equally, the requirements are prioritized based on their value to the customer. After they are prioritized and broken up into releases and/or iterations, the most important requirements, which are at the top of the list and ready for development, get developed first.
- Once the developer has picked up a user story to begin working on, he/she will then work directly with the user to *pull* more detail as needed to fill that requirement.

A story card is equivalent to a Kanban card in a manufacturing system and describes one particular feature that a user needs. As in the manufacturing system, physical cards may or may not be used—there are computerized tools that will automate this task and eliminate the use of physical story cards if desired. However, in many cases, physical cards may actually be used and put on a board; each developer picks up a card to start working on it, similar to the way a Kanban card works in a factory.

A Kanban development process is sometimes used as an alternative to Scrum in situations that need to be more reactive, such as managing a queue of customer service requests. In a Scrum process, there is some level of upfront planning to at least identify the items in the product backlog at a high level before the project starts, and additions to the backlog items are not allowed once an iteration has started. From that perspective, a Scrum process is only *partially* based on *pull*. At a high level, requirements are *pulled* through the process based on priority; however, within an individual iteration, once the iteration has started, requirements are *pushed* through the process.

A Kanban process is designed to be much more reactive and responsive to customer demand. The following table shows a summary of the differences between a Kanban development process and Scrum:[10]

	Scrum	Kanban
Primary Flow	Time-boxed iterations (sprints)	Continuous flow
Primary Metric	Velocity	Lead time
Work-in-Process (WIP)	WIP limited indirectly (per sprint)	WIP limited directly (per workflow state)
Addition of New Items	No new items added to the current sprint	Add new item as capacity is available
Roles	three prescribed roles (product owner, Scrum Master, team)	No prescribed roles
Status Board	Scrum board reset between sprints	Kanban board is persistent

The key difference is with regard to how the two processes manage flow.

- A Kanban process is *totally* pull and allows demand for new items to take place at any time, *including during the middle of a sprint* if capacity is available.
- A Scrum process is *mostly* pull. The product backlog is considered to be dynamic and can be adjusted as needed to meet customer needs within the constraints that (a) the capacity to handle the demand is fixed and (b) additions are not allowed once an iteration has started.

Flow

Flow is one of the most important lean principles to understand to maximize the efficiency of any process. There are a number of factors that contribute to maximizing flow, which include:

- Small batch sizes
- Just-in-time production
- Concurrent processing

Each of these topics is discussed in the following sections.

Small Batch Sizes

In a manufacturing process, it is well known that small batch sizes are much better for optimizing the flow of a process than large batch sizes. If large batch sizes are used, bottlenecks can easily develop

[10]Stephanie Stewart, Valpak Agile Overview PowerPoint Presentation

at various points in the process, and material winds up waiting at those bottleneck points, creating waste. There are at least a couple of types of *waste* associated with large batch sizes:

- Excess material inventory is used in the process, creating unnecessary inventory cost, space for storage, and additional handling costs. Mary Poppendiek uses an example of the construction of the Empire State Building in New York. The Empire State Building was the tallest building in the world and was built in a total of 20 months, including demolition of existing buildings and planning and design of the new building.[11]

  One of the most serious constraints was that there was only a limited amount of vacant real estate in the area to store the materials needed for the building. This meant that the flow of the project had to be very carefully planned. The building was built in iterations of a few floors at a time, and the arrival of material had to be scheduled meticulously to have just the right materials available at the right time to maximize the flow.

- If the inventory is perishable, it can go stale and become unusable. By *perishable,* I'm not necessarily referring to fruits and vegetables. Dell Computer is a good example. Dell builds systems for customers out of a variety of different components (disk drives, graphic cards, etc.), and those components become obsolete quickly and are constantly being replaced by newer versions. Using small batch sizes and building systems on demand as customers need them reduces the risk of winding up with too much obsolete inventory of components in the pipeline.[12]

The other major advantages of small batch sizes are[13]:

- It reduces the end-to-end cycle time (Little's Law of Queuing[14]).
- It makes waste a very hard problem to ignore as any waste in a small batch size system will cause much larger problems than when you've got inventory at hand to smooth it over. You then have to confront and fix the waste. The visual metaphor often employed is that a stream running low uncovers the rocks on its bed.

Attempting to define all the requirements for a product or application up front in a traditional development process is analogous to attempting to process large batch sizes in a manufacturing process. It's impossible to work on all the requirements at once, so bottlenecks develop at various points in the process and requirements wind up waiting to be processed. Having an excess of requirements sitting around waiting to be processed is similar to having excess inventory in a manufacturing process:

- There are handling costs associated with managing those requirements waiting to be processed—they have to be well documented and tracked or they may be forgotten and left out of the design. There is also a certain amount of overhead associated with managing changes to these requirements.

[11] Poppendiek, p. 102.
[12] Ibid, p. 12.
[13] Martin Burns, personal e-mail comments, on book review.
[14] "Principle: Little's Law," http://www.factoryphysics.com/Principle/LittlesLaw.htm.

- The requirements are also perishable—if they wait for a long time to be processed, they could easily become obsolete. If that happens, either someone winds up designing and building a product or application based on obsolete requirements, or unnecessary labor is consumed in redefining and rewriting the requirements.

An iterative development process is analogous to small production batch sizes in a manufacturing operation. By breaking up the requirements into the smallest possible units (user stories), the overall development process is likely to flow more smoothly and avoid the bottlenecks associated with traditional development processes. Of course, in actual practice, there are limits to how far it is practical to break down the requirements to optimize flow. For example:

1. *Requirements management considerations*. From a requirements management perspective, it may be necessary to group requirements into feature sets that are interrelated with each other. Those feature sets might be bigger than the effort that can be realized in a single iteration. That requires some compromises between:
   - An idealized approach where individual sets of requirements are completely processed immediately in each iteration; and
   - A more realistic hybrid approach where some of the requirements might be *pipelined* for processing and spread across more than one iteration.

     The idea of buffering some of the requirements that cannot be fulfilled immediately is called *story pipelining*.

> Story pipelining is often seen by purist Agile practitioners as strictly un-Agile as it violates the oft-held view that working software is the only thing that represents value and anything less is a cop-out. In practice, however, pipelining is often the best way to balance agility with the realities of real-world delivery constraints. You gain a measure of project progress that's still closer to real doneness in the eyes of the customer, you retain a lifecycle that encourages regular and frequent feedback, but you also recognize that complex software systems have a gestation period.
>
> Pipelining particularly recognizes requirements that are just really difficult to implement: thorny user interactions, challenging external systems dependencies, etc.[15]

2. *Testing/release and configuration management considerations*. Testing and release management considerations might require compromises from the ideal flow model:
   - Testing might want a functionality set to test against that's a relatively complete subset of functionality and will be stable for the duration of the test cycle.
   - Release and configuration management might have similar needs.

[15]Gottesman, Erik Comments on book review

Both of these issues can be overcome, but may require some rethinking of how the process works and very strong coordination of test planning with the rest of the development effort.

Just-in-Time Production

Having large quantities of raw materials waiting to be processed is inefficient because they become bottlenecks. A much more efficient process is based on just-in-time processing, when the raw materials arrive just at the right time that they are needed in production. A large business requirements document in a software development environment is equivalent to a large pile of raw materials in a manufacturing environment. Instead of developing large requirements documents that wait to be processed, it is generally more efficient to develop requirements just-in-time as needed in the software development process. For example:

- Early in the process, high-level requirements should be sufficient to do whatever level of planning is needed at that point.
- The elaboration of requirement details can be deferred until later in the process when those particular requirements are ready to enter development.

Concurrent Processing

Concurrent processing is another well-known way to improve flow in any process. In a manufacturing process, bottlenecks are much more likely to develop if there is only one path through the system and everything is sequential than if there are parallel paths available and some work can be done concurrently on different paths.

In a product development process, there are typically greater opportunities for concurrent engineering to improve the flow through the process. Here are a few examples:

- Requirements development can be overlapped with design instead of being sequential, and quality testing can also overlap with design instead of following it sequentially. This requires a much more collaborative, cross-functional approach to development, which can be difficult to achieve, but the potential payoff can be significant.
- Design teams can work on multiple iterations concurrently. This requires breaking up the design effort into iterations and requires some coordination among design teams:

> Concurrent engineering is especially useful when dealing with 'unprecedented' requirements . . . This can provide cover in situations where you have multiple integration approaches to choose from and don't know which is best (e.g., which will deliver the right performance, availability, or conform to other non-functional requirements). Similarly,

challenging UI problems are sometimes best tackled using concurrent engineering (e.g., prototyping multiple solutions and testing them with real, representative users) in a 'survival of the fittest' approach.[16]

Respect for people

In the early days of automotive manufacturing, processes were designed so that the people performing those processes did not require high levels of skill. An individual working on an assembly line could be assigned a small, repetitive task such as installing a tire on a car, which required only a minimum amount of skill and training. The primary requirement for higher levels of skill and training could be limited to a relatively few people who were responsible for designing and managing the overall process and training the workers to perform each task. There are several problems with that approach:

- No one really takes responsibility for the overall quality of the complete vehicle.
- This approach might rely heavily on quality control inspectors at the end of the line to try to find defects and send the vehicle back for rework if necessary.
- It can be a dehumanizing experience for anyone to perform that kind of limited, repetitive task.
- This approach doesn't take advantage of the complete range of skills and judgment of the people performing the tasks.

Manufacturing processes have long recognized the need to respect and empower the people performing the processes as much as possible. In a manufacturing process, having people take pride in workmanship is extremely important for achieving high levels of quality and productivity. The need for fully utilizing the capabilities of people and motivating them is even more significant in a software development process, where the overall effectiveness of the process is so critically dependent on the performance of the people. Both lean and agile methodologies seek to eliminate those problems by empowering individuals and the team as a whole to take responsibility for the overall quality of their work.

Many traditional development processes have been modeled on early manufacturing processes, where a process defines in detail the work to be done and how it should be done, and the process requires a lower level of skill to perform relatively well-defined tasks. Agile methodologies are generally much less well defined and rely heavily on the skill and training of the people performing the process to use appropriate levels of judgment and tailor them to a particular project, task, and business environment. That is a key reason why respect for people is so important in an agile environment.

[16] Ibid.

Perfection

The principle of perfection in lean manufacturing is similar to the TQM principles associated with ceasing reliance on inspection and ongoing continuous process improvement to remove defects. Defects in lean are seen as a major source of waste:

- It takes a lot of resources to inspect for defects, which wouldn't be necessary if the defects were eliminated at their sources.
- Rework and scrap can result from defective products if those defects aren't discovered until the product is at the end of the assembly line (or at the end of the development process in a software development environment).

Lean manufacturing also emphasizes continuous improvement to eliminate the waste caused by defects, just as TQM emphasizes it for improving product quality.

There is also a direct relationship with the principle of respect for people. In many cases, the people performing the process are the first ones to recognize opportunities for improvements in the process to prevent defects and/or to make the process more efficient. Unfortunately, many times they are not empowered to suggest or make those changes. Lean and agile methodologies recognize that and therefore are not rigidly defined or prescriptive—they provide some fundamental principles and practices that are common to most projects and are expected to be tailored to a given situation. And the people performing the process have a significant role in the design and management of the process. Naturally, it requires more skill to make good judgments about how to tailor a process to fit a business and project environment.

PRINCIPLES OF PRODUCT DEVELOPMENT FLOW

Don Reinertsen wrote a widely read book on the principles of product development flow, summarized here:[17]

1. *Economics: Take an economic view.* Many times there is a point of diminishing returns associated with improvements in a process—understanding the economic impact is a critical factor in optimizing a process. For example, it is *generally* best to defer decisions on product features as long as possible; however, some decisions should be made early and should not be significantly deferred because of their economic impact.

   Example: Increasing innovation should not be an end-in-itself and it reaches a point of diminishing returns at some point and begins to impact other proxy variables such as quality.
2. *Queues: actively manage queues.* Agile development processes are based more on a continuous flow process as opposed to heavily plan-driven processes that are more of a large-scale batch process. A continuous flow product development process operates most efficiently when queues are managed.

[17] Don Reinertsen, *The Principles of Product Development Flow* (Redondo Beach, CA: Celeritas Publishing, 2009).

Example: Developing requirements far-in-advance that sit in a queue waiting for development can be inefficient and wasteful because:

- The requirements may change prior to going into development and much of the effort involved in developing the requirements might have been wasted, and/or
- Speculation in the requirements that are done too far into the future can result in erroneous assumptions that make their way into development without being questioned.

3. *Variability: Understand and exploit variability.* Reducing variability will *many times* improve efficiency but that isn't always the case. For example:
   - Breaking up large requirements into smaller ones that are of a more uniform size reduces variability and can improve flow, however, at some point further attempts to reduce variability do not have economic value.
   - Forecasting errors are a major source of variability . . . we can reduce this variability by forecasting at shorter time horizons.
   - Design reuse reduces variability.
4. *Batch size: Reduce batch size.* Large batch sizes tend to cause bottlenecks and inhibit flow. Reducing the batch size by breaking up requirements into small, independent user stories can significantly improve flow.

   Examples of batch size inefficiencies include:
   - Project scope: More is taken on in a single project than is truly necessary.
   - Project funding: The entire project is conceived and funded as a single large batch proposal.
   - Requirements definition: the tendency to define 100% of the requirements before the project starts
5. *WIP constraints: Apply WIP constraints.* Use work in process (WIP) constraints to manage overall flow. For example,
   - Control the number of projects taken on at any one time to avoid oversaturating development resources.
   - Use specialized resources wisely to maximize their impact on overall flow.
6. *Control flow under uncertainty: Cadence and synchronization.* Reinertsen[18] defines cadence as follows:

> Cadence is the use of a regular predictable rhythm within a process. This rhythm transforms unpredictable events into predictable events. It plays an important role in preventing variability from accumulating in a sequential process . . .

[18] Ibid

Having a repeatable cadence improves the efficiency of the product development process and allows synchronizing a predictable development process with a much more unpredictable flow of requirements. Examples of the use of synchronization include:

- Concurrent development on multiple paths at the same time
- Concurrent testing of multiple subsystems

7. *Fast feedback: Get feedback as fast as possible*. Fast feedback can lower the expected loss by truncating unproductive paths more quickly or raise the expected gain because we can exploit an emergent opportunity by rapidly redirecting resources.

   Fast feedback, combined with selecting appropriate measures of performance, enables rapid learning and ongoing continuous improvement.
8. *Decentralized control: Decentralize control*. The final principle that Reinertsen has identified deals with decentralized control:

> Sophisticated military organizations can provide very advanced models of centrally coordinated, decentralized control. There is an impression that military organizations seek robotic compliance from subordinates to the orders of superiors. In reality, the modern military focuses on responding to a fluid battlefield, rather than executing a predetermined plan. It views war as a domain of inherent uncertainty, where the side that can best exploit uncertainty will win.[19]

SUMMARY OF KEY POINTS

Systems Thinking

1. Systems thinking is very important in order to see agile principles and practices in a holistic sense and in the context of how they fit with the overall business objectives of an enterprise. In the context of agile, *systems thinking* means understanding the principles behind the methodology rather than just focusing on the mechanics of how the methodology works and understanding how those principles interact with the overall project and business environment that they are part of.

Influence of Total Quality Management (TQM)

2. TQM and the thinking behind it revolutionized the quality and competitiveness of the automotive industry. TQM taught us to:

   - *Eliminate defects at the source:* Develop a more systemic approach to designing quality into the process that produces products to eliminate the defects at the source, rather than constantly finding and fixing the same or similar defects over and over again.

[19] Ibid.

- *Recognize the human aspects of quality:* Recognize the human aspects that are essential to build quality, such as engaging people at all levels so that they feel responsibility and ownership for the quality of the overall products they produce and empowering them to recognize and suggest opportunities for improvement in the process.
- *Develop a cross-functional approach:* Break down barriers between departments, eliminate conflicting goals among organizations, and develop a much more integrated cross-functional approach that leads to much higher levels of productivity and efficiency, together with a more collaborative approach to quality and process improvement.
- *Recognize the importance of leadership:* Eliminate traditional command-and-control management in favor of inspirational leadership to empower people and to help them see the higher-level purpose and vision for their work.
- *Strive for ongoing continuous improvement:* Commit to an ongoing continuous improvement effort to constantly find opportunities to improve processes.

Influence of Lean Manufacturing

3. The concept of lean originated in manufacturing and has had a significant impact on many industries. The focus of lean manufacturing is on elimination of waste and improving operational efficiency rather than simply improving quality; however, lean does recognize quality defects as an important form of waste that should be eliminated. Lean taught us:

- *Customer value focus:* Focus on producing customer value and eliminate all unnecessary tasks that do not add value to the customer.
- *Map the value stream:* Map the value stream to understand the process flow and identify any opportunities to eliminate non–value-added steps.
- *Pull:* Use a pull approach rather than a push approach to plan and manage production capacity to meet demand.
- *Flow:* Use the principles of flow, such as just-in-time processing, to optimize the efficiency of the overall process.
- *Respect for people:* Recognize and be sensitive to the human aspects of quality. People need to be respected and properly motivated to develop high-performance teams that produce very-high-quality products.
- *Perfection:* Use a systemic approach to identify the source of waste and defects in a process and use a continuous incremental improvement approach to perfect the process.

These same principles are adaptable to a software development environment.

Principles of Product Development Flow

4. A traditional management approach is heavily based on managing and controlling the *structure* of a project with such things as work breakdown structures, Pert charts, and Gantt charts. An agile project is based on a much more fluid and adaptive process where structure is much less important and managing the flow of items through the process is much more important.

5. Understanding the principles of product development flow is very important to optimize the efficiency of an agile project. The key principles of product development flow outlined by Don Reinertsen are:
   - Economics: Take an economic view.
   - Queues: Actively manage queues.
   - Variability: Understand and exploit variability.
   - Batch size: Reduce batch size.
   - WIP constraints: Apply WIP constraints.
   - Cadence and synchronization: Control flow under uncertainty.
   - Fast feedback: Get feedback as fast as possible.
   - Decentralized control: Decentralize control.

DISCUSSION TOPICS

Systems Thinking

1. Discuss an example of a problematic situation where you might have used systems thinking to analyze the situation and determine an appropriate solution.

Influence of Total Quality Management

2. Discuss an example of a problematic situation where the TQM principles might have been used to improve the overall quality of the work. What were some of the issues that impacted the quality of the overall product or service that the company produced? How could the impact of those issues been reduced or eliminated?
3. How do you think that an understanding of the principles behind Total Quality Management might influence an agile project management approach?

Influence of Lean Manufacturing

4. Discuss an example of a situation where lean thinking could have been used to improve the overall efficiency of the process. What was some of the waste involved in the process? How could it have been reduced or eliminated? What principles of lean did you use to make that assessment?
5. How do you think that an understanding of lean manufacturing principles might influence an agile project management approach?

Principles of Product Development Flow

6. Analyze the following situations and identify the appropriate principles of product development that you might be important to consider in defining an appropriate solution to each:
   - A project is taking much longer than originally anticipated: it was originally planned to take only 6 months, it has now gone on for over 18 months, and there still doesn't seem to be an end in sight.

- Projects are being delayed because users are frequently changing requirements in the middle of the project.
- The quality assurance organization is becoming a bottleneck and scheduled releases are being delayed waiting for QA testing.
- A company has just finished a massive implementation of agile throughout the entire company but senior executives are not satisfied with the results. They feel like they have lost some visibility into projects and are not sure the projects are well-aligned with their business goals.
- An agile development team is repeatedly missing its sprint goals because it has overcommitted the amount of work that can be done in a sprint.
- The company CEO insists on personally making decisions in what he considers to be the most critical projects to the company's success.