



Jack
be Agile

Jack
be Lean



SUSTAINABLE PRODUCT DEVELOPMENT
THE LEAN-AGILE CONNECTION
by Alan Shalloway

LEAN SOFTWARE DEVELOPMENT AND AGILE SOFTWARE DEVELOPMENT ARE TWO APPROACHES ON THE MINDS OF MANY WHO WANT TO BECOME MORE EFFECTIVE SOFTWARE DEVELOPERS. LEAN? AGILE? ARE THEY SIMILAR OR DIFFERENT? DO THEY COMPETE OR COMPLEMENT EACH OTHER?

Similar...

While microeconomics and macroeconomics focus on different levels of behavior in an economy, they are interdependent and complement each other. In this same way, agile and lean offer different—yet complementary—perspectives on software development. Agile is more bottom-up and team-centered; lean is more top-down and enterprise-centered. Depending on your situation, you may choose to lead with one or the other, but ultimately, a combination is most effective. Comparing and contrasting agile with lean will help you understand the contribution that each can make to your success.

Agile Software Development

The focus of agile software development is on building software. Its scope begins with the approval of a project and ends with final deployment. The heart of agile is the team; the heartbeat is the iteration.

A number of agile methods are popular today—Scrum, Extreme Programming (XP), Crystal, and feature-driven development. Agile methods focus on building the features most important from the customer's viewpoint while creating a supportive working environment for the team. All agile methods include these four key elements:

- Iterative development (building software in increments of one to four weeks)
- Re-evaluating the situation after each iteration
- Close contact with the customer to guide the team
- A team-centric approach that encourages teams to make decisions about how the software will be built

Developing software in increments means that the software is continuously being modified. Therefore, many agile methods require or strongly recommend using automated acceptance and unit testing to minimize regression defects.

In many ways, agile arose from the challenges of other methods. Previously, it was thought that the ideal way to build software was by completely defining what was needed before building anything. The team then would create an architecture; design and write the code; and, finally, verify that it worked. This approach was based on the notion that changes were expensive, that clarity was the best way to prevent them, and that this clarity could be obtained through analysis. But there are difficulties with this approach.

At the beginning of most projects, customers rarely are sure of the system requirements. They often ask for functions or attributes that they believe will be useful but which turn out not to be. Another difficulty is that customers understand their own needs better once you show them what they have requested. The non-agile approach gives us only one shot at getting the analysis and design right. Discovering requirements late in the project often results in shoe-horning in features that the system wasn't designed to accommodate. Including less-than-useful features increases the complexity of the software and ultimately the cost to maintain and extend it.

The usual separation of groups by roles, combined with relying on inter-group communication through written documents, further compounds these problems. Written documentation is wonderful if you are trying to communicate to thousands of people. However, written communication is not as effective as interactive communication. If it were, universities wouldn't be needed—we'd learn everything from books.

Lean Software Development

Lean software development is based on lean thinking—the approach that was created and honed by Toyota in its manufacturing and product development. Many authors describe lean thinking by discussing the practices of Toyota, inferring the principles underlying those practices, and then reapplying those principles to the arena in which the authors are working. In this article, we'll start with the principles themselves, as these are self-evident in the software world. We will consider what practices lean thinking infers and examine why they are useful.

The foundation of lean thinking is:

- Respect people.
- Continuously improve your process.
- Respect knowledge.
- Add value as quickly as possible to your customer while retaining the ability to add value quickly in the future.

This may sound all well and good but, as someone in one of my classes once asked, “What do we actually do?” In their book *Lean Thinking*, James Womack and Daniel Jones explain the goal of lean as a way of creating “fast-flexible-flow” from idea to implementation. Anything that impedes the ability to do this is waste. In the software world, these wastes are usually due to delays, rework, and multitasking. In their book *Imple-*

menting *Lean Software Development: From Concept to Cash*, Mary and Tom Poppendieck are more explicit in describing “seven principles” of lean software development:

1. **Eliminate waste**—Don’t build things you don’t need. Focus on small units of complete functionality to maximize efficiency. Use effective processes to accomplish your delivery goals.
2. **Build quality in**—Don’t create the bugs that developers spend so much time finding and removing. Testing must play an active role in improving the process being fol-

lowed—not merely fixing bugs at the end. Test-and-fix cycles are an indicator of a poor process.

4. **Defer commitment**—Defer irreversible decisions as long as possible—but not later than is responsible. In software development, this means avoiding constraining a system too early with a comprehensive but inflexible architecture. It also means not accepting all of what the customer says early on as sacrosanct, but giving him an option to change his mind after seeing early versions of the system.
5. **Deliver fast**—We must not lose sight of our goal—deliver value

actually constructing it. That’s product development.

are interested in maximizing the value delivered to the customer—not merely maximizing any one (or even all) of the steps in the process.

These principles typically work together synergistically. An example is developers and testers working together to run validation tests quickly after the code has been developed (or even concurrently with development). This contrasts with focusing on maximizing the creation of code and then maximizing the running of tests. Focusing on optimizing each of these steps often results in delays between them, which actually hurt the development process rather than improving it. Note the connection here with fast-flexible-flow. Focusing on local-optimizations causes delays between steps, which impedes flow. Lean says optimize the whole, eliminate waste, and deliver fast.



“Focusing on optimizing each of these steps often results in delays between them, which actually hurt the development process rather than improving it.”

3. **Create knowledge**—Software development is not like engineering; it is not like construction; it is product development. In *Product Development for the Lean Enterprise*, Michael Kennedy writes, “A useful definition [of product development] is that it is the collective activities, or system, that a company uses to convert its technology and ideas into a stream of products that meet the needs of customers and the strategic goals of the company.” Most of the time we spend in software development is in discovering what the customer wants and how we will create it. Only 10 to 20 percent of the time is spent in

quickly to the customer (while retaining the ability to add value quickly in the future). This becomes a strong competitive advantage—quick to market, customer loyalty, lower costs.

6. **Respect people**—We must create processes that support our people. Effective processes cannot be dictated from outside the team, either by management or by consultants. Only the team truly knows itself and its capabilities. W. Edwards Deming was a strong proponent of this fundamental principle on which lean is based (see the StickyNotes for more information).
7. **Optimize the whole**—Focus on the entire chain of events of building software. This is equivalent to fast-flexible-flow in manufacturing. We

...But Not the Same

Agile and lean differ in the following ways:

Scope of the software development cycle being addressed

The agile methods—in particular Scrum and XP—describe how to run software projects from the point of inception to deployment. Their focus is on the development project and the team doing the development work. Lean takes a broader view. It includes processes within which projects are spawned. It looks beyond the project to how it fits within the enterprise. Lean is not limited to the project’s value to the business but looks at the entire process of selecting projects. Lean creates multiple opportunities for improvement that are outside the scope of agile methods. Lean thinking:

- Ensures that projects are spawned and managed in a cohesive way
- Connects projects back to the business value they are to provide, creating a focus on the products of the company and not just on its de-



A project that is delayed
six months in initiation
causes the same
problems for its clients
as a project that is
delayed six months in
implementation.”

velopment projects

- Manages resources across projects so that knowledge is not lost
- Eliminates waste caused by poor collaboration of teams and tight dependencies between them

The bottom line is that agile looks to assist the team to work in the best possible way within its context. However, agile ignores potential improvements to this context—improvements that may actually be more important than improving the software development process itself.

Why is this important? Because many of the significant impediments to effective, efficient software development have more to do with the organization within which the team exists than with the activities of the team. For example, we had a client whose development team was constantly being interrupted with emergencies because its customers did not properly prepare their servers before product deployment. This occurred because the sales team focused on “making the sale” and

not on helping customers complete the pre-deployment checklist. Once the sales team understood lean’s “optimize the whole” concept and realized that ensuring smooth installations would result in additional sales, the development team’s life became much easier, and its throughput and morale improved as well. Lean looks at processes outside the team to try to optimize the whole “value stream” that creates product and delivers it to the customer.

The broader view of lean has us focus more on the product rather than just the project. A product is something that provides value to the business by providing value to the customer. It is what the customer wants, values, and receives. Agile focuses on individual projects; lean concentrates on the entire product.

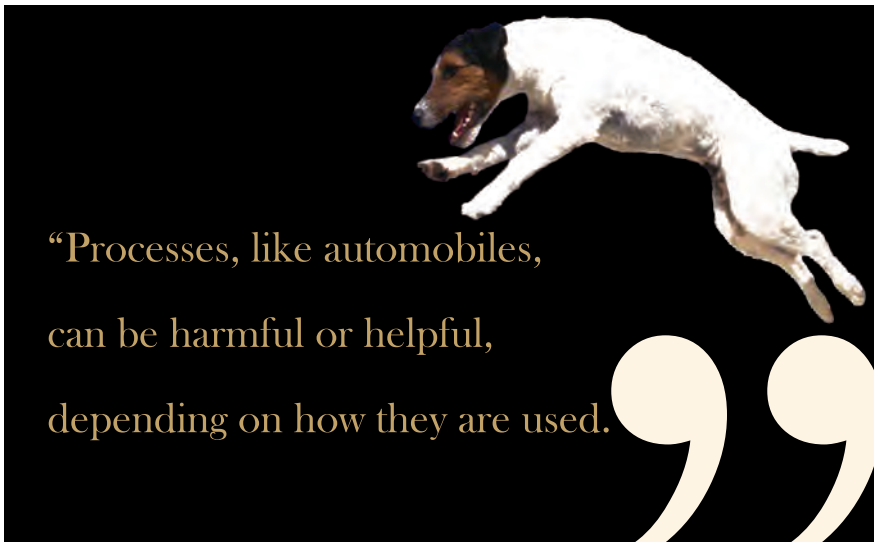
A project-centric focus tends to result in shorter-term decisions—what is best for this project, for this team. Often, once a project is finished, teams are disbanded and must be reformed when the next project begins. Not only is this expensive,

but lessons-learned and process improvements tend not to be passed along, or at least not as effectively as if the team stayed together for the entire product. Such inefficiency harms the ability of the business to deliver value to the customer as quickly as possible.

Lean’s view focuses on respecting people and the knowledge and skill they possess. Its product-centric view encourages management to look at the bigger picture.

Team versus enterprise focus

Agile focuses on the team. The team focuses on the project. The project does not exist until it has been initiated. Typically, vital product and project decisions are made before initiation—a period that agile methods ignore. One of the fundamental principles of lean software development is to “optimize the whole.” This means to look at how to improve the entire process—not just individual phases—of building software. A project



“Processes, like automobiles,
can be harmful or helpful,
depending on how they are used.”

that is delayed six months in initiation causes the same problems for its clients as a project that is delayed six months in implementation. Yet, agile methods don't address the first situation; lean does.

Additionally, in many software development organizations, the management metrics of the different roles involved often get in the way of having an effective team. Business analysts, developers, testers, and project managers all must be measured in ways that create synergy with one another rather than win/lose competition or a “not my job” mentality. Agile software development addresses this with the notion of the “team swarm,” in which a cross section of people works together to complete a story or a task. However, if performance metrics focus on individuals rather than teams, the organization will work against the team. Lean thinking addresses structural change at the organizational level. It does this by providing management with the insight necessary to understand the entire process—from inception of idea to implementation—instead of optimizing individual role efforts.

Attitude about process

“Process” has gotten a bad name in the software industry—and it often is deserved. But just because individuals abuse or misuse process ideas does not mean that that process is inherently bad. For example: Something kills more than one half million people a year. It injures almost 40 million people a year. It is a major contributor to air pollution which

kills and injures millions more. Would you say this is a bad thing? Of course, but I'd be willing to bet you have one of these things. In fact, it's likely that every adult in your household has one. What is it? An automobile.

Processes, like automobiles, can be harmful or helpful, depending on how they are used.

In software development, harmful processes are those that strip energy and effort from the development team but do not add value. Agile development sprang from a reaction to ineffective processes. New processes aren't necessarily bad—“a little revolution now and then is a healthy thing” (Captain Marko Ramius, *The Hunt for Red October*); however, an overreaction can inhibit positive improvement once things have returned to a stable state.

Lean starts from the position that process exists to help people work together but must always be improved. Lean does not tolerate workarounds. The most direct example of this in software development is improving process to eliminate the creation of defects in development instead of fixing them at the end. In this regard, many agile practices are implementations of lean principles; automated testing and test-driven development are two examples. The major difference is that lean focuses on principles—eliminate waste, don't tolerate workarounds, eliminate delays—that can be implemented within practices. Agile methods limit process improvement to having teams adjust given, pre-defined processes to their projects.

Decision-making and process improvement tools

Agile's focus on the team does not provide team members with decision-making and process improvement tools. In addition, agile's somewhat “anti-process” view has biased it away from anything that can be turned into a heavy-weight process. This, unfortunately, has left teams on their own to figure out how to improve.

Lean, on the other hand, incorporates a number of formal tools for helping teams and organizations break through to improve processes, including:

- **Value stream mapping**—Describes flows of information and material in the overall development process. It helps expose the process challenges and the wastes in time and information. It is an essential first step in process improvement and waste elimination.
- **Profit and loss**—Uses hypothetical internal profit-and-loss statements to assess how decisions in product timing and mix affect resources, returns, and costs.
- **5S**—Simplifies the work environment and information structure so that things that are needed can be found. The steps of 5S are sort, set in order, systematize, standardize, and sustain. Its focus is on identifying effective practices, following them, and integrating them into the team.
- **Kaizen**—As defined on wikipedia.org (see the StickyNotes for a link), kaizen is a daily activity whose purpose goes beyond improvement. It is a process that, when performed correctly, humanizes the workplace, eliminates difficult work (both mental and physical), and teaches people how to do rapid experiments using the scientific method and how to learn to find and eliminate waste in business processes.
- **Retrospection**—Agile advocates the use of retrospectives in which the team examines what it did so it can better adjust what it will do in the future. Lean emphasizes retrospectives at more levels and at a

greater frequency.

- **Standard work**—Lean takes the agile concept of standard work (such as the daily meeting) across value stream management, recognizing that every level has a part to play in improvement. Standard work ensures that the routine work at every level always is done. Lean follows up with routine assessments to examine how things are being done.

Where Should You Start?

Both lean and agile have much to contribute to effective software development. So, where to start? The answer depends upon who is doing the starting.

If your development teams are somewhat independent, and if teams generally remain together over multiple projects, agile often provides a good place to start. You should expect to improve development practices very quickly. However, most software development organizations do not meet these prerequisites. In that case, starting with lean software de-

velopment offers a larger context for process improvement. Lean can tackle issues beyond an individual team's performance. However, lean also requires greater management support and commitment.

Agile methods, on the other hand, can be adopted by a team without requiring management buy-in. In fact, once agile methods have been adopted, the impediments to the team often can be presented to management as motivation for proceeding with lean thinking. If management is not ready or needs to “put a toe in the water first,” then agile is a good way to begin—just don't stop there!

Conclusion

Lean and agile are compatible, each contributing important elements to the overall software development process. Lean focuses on delivering value to the business in a sustainable way by focusing on eliminating waste and by improving processes that support the development team. Lean involves a thought process that incorporates a number of process

evaluation tools. Lean implies the need for an agile process. The term “lean-agile” is sometimes used to mean an agile process that takes advantage of lean thinking. To be truly effective, one must adopt both. **{end}**

Alan Shalloway is the founder and CEO of Net Objectives. He is the primary author of Design Patterns Explained: A New Perspective on Object-Oriented Design and is writing a book on lean anti-patterns. Alan is a certified ScrumMaster and has a master's degree in computer science from MIT.



Sticky Notes

For more on the following topics go to www.StickyMinds.com/bettersoftware.

- W. Edwards Deming
- Kaizen

PNSQC 2007

1983-2007
CELEBRATING 25 YEARS OF QUALITY

ANNOUNCING THE 2007 DISTINGUISHED SPEAKER LIST

BUILDING FOR A BETTER FUTURE

On **October 8 – 10, 2007**, at the Oregon Convention Center, PNSQC will celebrate 25 years of bringing together a community of software professionals keenly interested in software quality. The list of distinguished speakers for this event includes:

KEYNOTES — **Johanna Rothman**, *Schedule Games: Recognizing and Avoiding the Games We Play*, and **Hugh Thompson**, *Hunting Security Vulnerabilities: The Illustrated Guide*.

INVITED TALKS — **Jennitta Andrea**, *Functional Tests as Effective Requirements Specifications*; **Lisa Crispin**, *Transitioning to Agile: Tips for Test Teams*; **Dale Emery**, *Resistance as a Resource*; and **Dot Graham**, *Cognitive Illusions in Development and Testing*.

WORKSHOPS — **Jennitta Andrea**, *Mastering Agile Requirements: Principles and Processes*; **Lisa Crispin**, *Getting Traction on Test Automation the Agile Way*; **Dale Emery**, *Mastering the Art of Change*; **Dot Graham**, *Agile Inspection*; and **Johanna Rothman** *Using Lifecycles to Design Your Project*

TO LEARN MORE
visit www.pnscq.org

25TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE