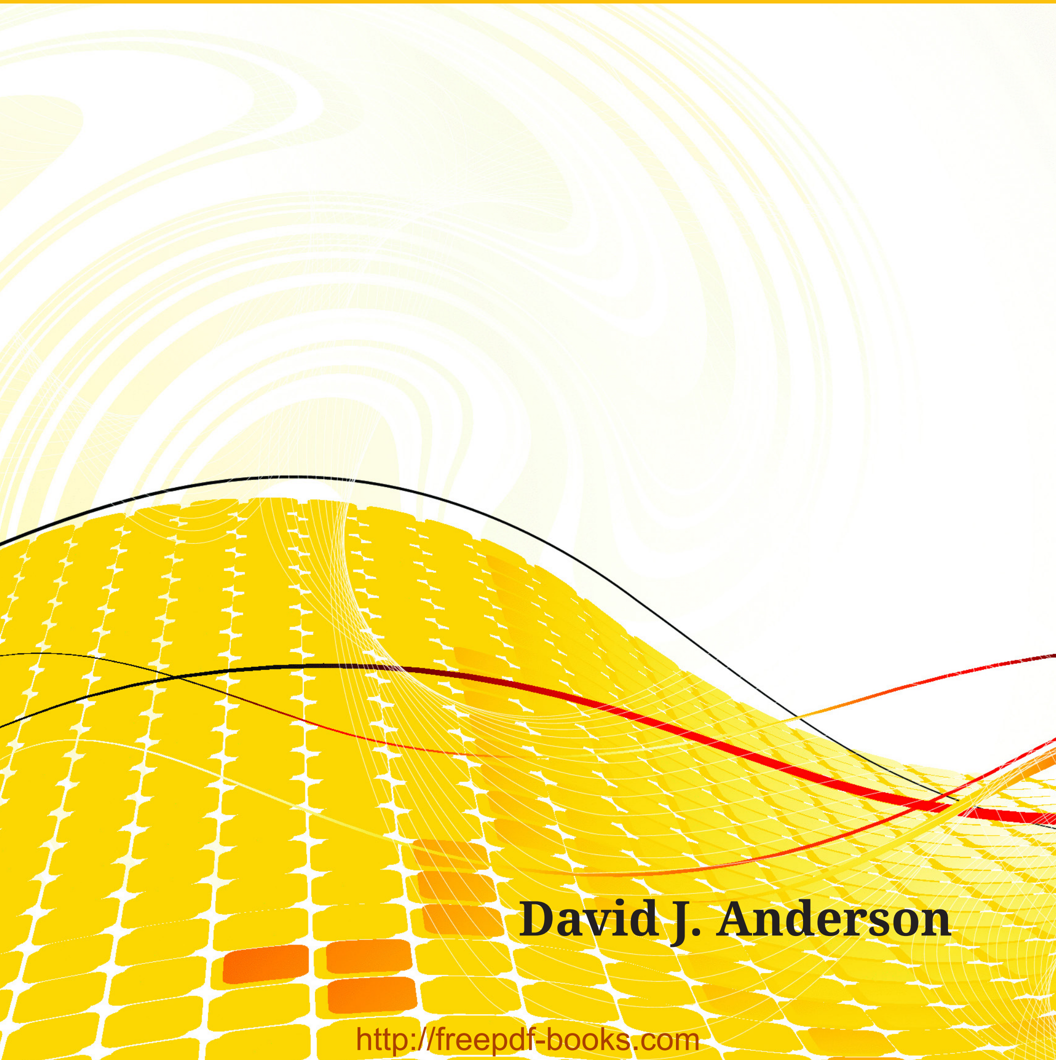
LKU | Lean-Kanban University

Lean Software Development



David J. Anderson

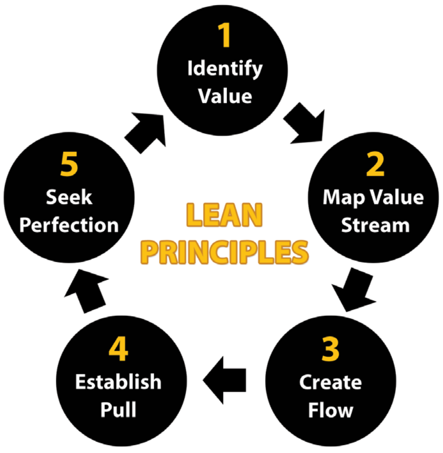
Lean-Kanban University, Seattle, WA

www.leankanbanuniversity.com

The term *Lean Software Development* was first coined as the title for a conference organized by the ESPRIT initiative of the European Union, in Stuttgart Germany, October 1992. Independently, the following year, Robert “Bob” Charette in 1993 suggested the concept of “Lean Software Development” as part of his work exploring better ways of managing risk in software projects. The term “Lean” dates to 1991, suggested by James Womack, Daniel Jones, and Daniel Roos, in their book The Machine That Changed the World: The Story of Lean Production [1] as the English language term to describe the management approach used at Toyota. The idea that Lean might be applicable in software development was established very early, only 1 to 2 years after the term was first used in association with trends in manufacturing processes and industrial engineering.

In their 2nd book, published in 1995, Womack and Jones [2] defined five core pillars of Lean Thinking. These were:

- Value
- Value Stream
- Flow
- Pull
- Perfection



This became the default working definition for Lean over most of the next decade. The pursuit of perfection, it was suggested, was achieved by eliminating waste. While there were 5 pillars, it was the 5th one, pursuit of perfection through the systemic identification of wasteful activities and their elimination, that really resonated with a wide audience. Lean became almost exclusively associated with the practice of elimination of waste through the late 1990s and the early part of the 21st Century.

The Womack and Jones definition for Lean is not shared universally. The principles of management at Toyota are far more subtle. The single word “waste” in English is described more richly with three Japanese terms:

- **Muda** – literally meaning “waste” but implying non-value-added activity
- **Mura** – meaning “unevenness” and interpreted as “variability in flow”
- **Muri** – meaning “overburdening” or “unreasonableness”

Perfection is pursued through the reduction of non-value-added activity but also through the smoothing of flow and the elimination of overburdening. In addition, the Toyota approach was based in a foundational respect for people and heavily influenced by the teachings of 20th century quality assurance and statistical process control experts such as W. Edwards Deming.

Unfortunately, there are almost as many definitions for Lean as there are authors on the subject.

Since 2007, the emergence of Lean as a new force in the progress of the software development profession has been focused on improving flow, managing risk, and improving (management) decision making. Kanban has become a major enabler for Lean initiatives in IT-related work.

Lean and Agile

Bob Charette was invited but unable to attend the 2001 meeting at Snowbird, Utah, where the Manifesto for Agile Software Development [3] was authored. Despite missing this historic meeting, Lean Software Development was considered as one of several Agile approaches to software development. Jim Highsmith dedicated a chapter of his 2002 book [4] to an interview with Bob about the topic. Later, Mary & Tom Poppendieck went on to author a series of 3 [5,6,7] books. During the first few years of the 21st Century, Lean principles were used to explain why Agile methods were better. Lean explained that Agile methods contained little “waste” and hence produced a better economic outcome. Lean principles were used as a “permission giver” to adopt Agile methods.

Lean Beyond Agile

In recent years, Lean Software Development has really emerged as its own discipline related to, but not specifically a subset of the Agile movement. This evolution started with the synthesis of ideas from Lean Product Development and the work of Donald G. Reinertsen [8,9] and ideas emerging from the non-Agile world of large scale system engineering and the writing of James Sutton and Peter Middleton [10]. I also synthesized the work of Eli Goldratt and W. Edwards Deming and developed a focus on flow rather than waste reduction [11]. At the behest of Reinertsen around 2005, I introduced the use of kanban systems that limit work-in-progress and "pull" new work only when the system is ready to process it. Alan Shalloway added his thoughts on Lean software development in his 2009 book on the topic [12]. Since 2007, the emergence of Lean as a new force in the progress of the software development profession has been focused on improving flow, managing risk, and improving (management) decision making. Kanban has become a major enabler for Lean initiatives in IT-related work. It appears that a focus on flow, rather than a focus on waste elimination, is proving a better catalyst for continuous improvement within knowledge work activities such as software development.

Defining Lean Software Development

Defining Lean Software Development is challenging because there is no specific Lean Software Development method or process. Lean is not an equivalent of Personal Software Process, V-Model, Spiral Model, EVO, Feature-Driven Development, Extreme Programming, Scrum, or Test-Driven Development. A software development lifecycle process or a project management process could be said to be "lean" if it was observed to be aligned with the values of the Lean Software Development movement and the principles of Lean Software Development. So those anticipating a simple recipe that can be followed and named Lean Software Development will be disappointed. You must fashion or tailor your own software development process by understanding Lean principles and adopting the core values of Lean.

There are several schools of thought within Lean Software Development. The largest, and arguably leading, school is the Lean-Systems Society which is based on the Kanban Method. Mary and Tom Poppendieck's work stands separately, as does the work of Craig Larman, Bas Vodde [13,14], and, most recently, Jim Coplien [15]. This article seeks to be broadly representative of the Lean Systems Society (LSS) viewpoint and to provide a synthesis and summary of the LSS ideas.

Values

The Lean Software & Systems Consortium (reorganized in 2012 as the Lean Systems Society [16]) published its values and principles at the 2011 Lean Software & Systems Conference [17]. It listed the following values:

- Accept the human condition
- Accept that complexity & uncertainty are natural to knowledge work
- Work towards a better Economic Outcome
- While enabling a better Sociological Outcome
- Seek, embrace & question ideas from a wide range of disciplines
- A values-based community enhances the speed & depth of positive change

Accept the Human Condition

Knowledge work such as software development is undertaken by human beings. We humans are inherently complex and, while logical thinkers, we are also led by our emotions and some inherent animalistic traits that can't reasonably be overcome. Our psychology and neuropsychology must be taken into account when designing systems or processes within which we work. Our social behavior must also be accommodated. Humans are inherently emotional, social, and tribal, and our behavior changes with fatigue and stress. Successful processes will be those that embrace and accommodate the human condition rather than those that try to deny it and assume logical, machine-like behavior.

Accept that Complexity & Uncertainty are Natural to Knowledge Work

The behavior of customers and markets are unpredictable. The flow of work through a process and a collection of workers is unpredictable. Defects and required rework are unpredictable. There is inherent chance or seemingly random behavior at many levels within software development. The purpose, goals, and scope of projects tend to change while they are being delivered. Some of this uncertainty and variability, though initially unknown, is knowable in the sense that it can be studied and quantified and its risks managed, but some variability is unknowable in advance and cannot be adequately anticipated. As a result, systems of Lean Software Development must be able to react to unfolding events, and the system must be able to adapt to changing circumstances. Hence any Lean Software Development process must exist within a framework that permits adaptation (of the process) to unfolding events.

Work Towards a Better Economic Outcome

Human activities such as Lean Software Development should be focused on producing a better economic outcome. Capitalism is acceptable when it contributes both to the value of the business and the benefit of the customer. Investors and owners of businesses deserve a return on investment. Employees and workers deserve a fair rate of pay for a fair effort in performing the work. Customers deserve a good product or service that delivers on its promised benefits in exchange for a fair price paid. Better economic outcomes will involve delivery of more value to the customer, at lower cost, while managing the capital deployed by the investors or owners in the most effective way possible.

Enable a Better Sociological Outcome

Better economic outcomes should not be delivered at the expense of those performing the work. Creating a workplace that respects people by accepting the human condition and provides systems of work that respect the psychological and sociological nature of people is essential. Creating a great place to do great work is a core value of the Lean Software Development community.

Principles

The Lean Software & Systems community seems to agree on a few principles that underpin Lean Software Development processes.

- Follow a Systems Thinking & Design Approach
- Emergent Outcomes can be Influenced by Architecting the Context of a Complex Adaptive System
- Respect People (as part of the system)
- Use the Scientific Method (to drive improvements)
- Encourage Leadership
- Generate Visibility (into work, workflow, and system operation)
- Reduce Flow Time
- Reduce Waste to Improve Efficiency

Follow a Systems Thinking & Design Approach

This is often referred to in Lean literature as "optimize the whole," which implies that it is the output from the entire system (or process) that we desire to optimize, and we shouldn't mistakenly optimize parts in the hope that it will magically optimize the whole. Most practitioners believe the corollary to be true, that optimizing parts (local optimization) will lead to a suboptimal outcome.

A Lean Systems Thinking and Design Approach requires that we consider the demands on the system made by external stakeholders, such as customers, and the desired outcome required by those stakeholders. We must study the nature of demand and compare it with the capability of our system to deliver. Demand will include so-called "value demand," for which customers are willing to pay, and "failure demand," which is typically rework or additional demand caused by a failure in the supply of value demand. Failure demand often takes two forms: rework on previously delivered value demand and additional services or support due to a failure in supplying value demand. In software development, failure demand is typically requests for bug fixes and requests to a customer care or help desk function.

Capability is 95% influenced by system design, and only 5% by the variance in performance of individuals.

A systems design approach requires that we also follow the Plan-Do-Study-Act (PDSA) approach to process design and improvement. W. Edwards Deming used the words "study" and "capability" to imply that we study the natural philosophy of our system's behavior. This system consists of our software development process and all the people operating it. It will have an observable behavior in terms of lead time, quality, quantity of features or functions delivered (referred to in Agile literature as "velocity"), and so forth. These metrics will exhibit variability and, by studying the mean and spread of variation, we can develop an understanding of our capability. If this is mismatched with the demand and customer expectations, then the system will need to be redesigned to close the gap.

Deming also taught that capability is 95% influenced by system design, and only 5% by the variance in performance of individuals. In other words, we can respect people by not blaming them for a gap in capability compared to demand and by redesigning the system to enable them to be successful.

To understand system design, we must have a scientific understanding of the dynamics of system capability and how it might be affected. Models are developed to predict the dynamics of the system. While there are many possible models, several popular ones are in common usage: the understanding of economic costs; so-called transaction and coordination costs that relate to production of customer-valued products or services; the Theory of Constraints – the understanding of bottlenecks; and The Theory of Profound Knowledge – the study and recognition of variability as either common to the system design or special and external to the system design.

Emergent Outcomes can be Influenced by Architecting the Context for a Complex Adaptive System

Complex systems have starting conditions and simple rules that, when run iteratively, produce an emergent outcome. Emergent outcomes are difficult or impossible to predict given the starting conditions. The computer science experiment "The Game of Life" is an example of a complex system. A complex adaptive system has within it some self-awareness and an internal method of reflection that enables it to consider how well its current set of rules is enabling it to achieve a desired outcome. The complex adaptive system may then choose to adapt itself – to change its simple rules – to close the gap between the current outcome and the desired outcome. The Game of Life adapted such that the rules could be re-written during play would be a complex adaptive system.

In software development processes, the "simple rules" of complex adaptive systems are the policies that make up the process definition. The core principle here is based in the belief that developing software products and services is not a deterministic activity, and hence a defined process that cannot adapt itself will not be an adequate response to unforeseeable events. Hence, the process designed as part of our system thinking and design approach must be adaptable. It adapts through the modification of the policies of which it is made.

The Kanban approach to Lean Software Development utilizes this concept by treating the policies of the Kanban pull system as the "simple rules," and the starting conditions are that work and workflow is visualized, that flow is managed using an understanding of system dynamics, and that the organization uses a scientific approach to understanding, proposing, and implementing process improvements.

Complex systems have starting conditions and simple rules that, when run iteratively, produce an emergent outcome. Emergent outcomes are difficult or impossible to predict given the starting conditions.

Respect People

The Lean community adopts Peter Drucker's definition of knowledge work that states that workers are knowledge workers if they are more knowledgeable about the work they perform than their bosses. This creates the implication that workers are best placed to make decisions about how to perform work and how to modify processes to improve how work is performed. So the voice of the worker should be respected. Workers should be empowered to self-organize to complete work and achieve desired outcomes. They should also be empowered to suggest and implement process improvement opportunities or "kaizen events" as they are referred to in Lean literature. Making process policies explicit so that workers are aware of the rules that constrain them is another way of respecting them. Clearly defined rules encourage self-organization by removing fear and the need for courage. Respecting people by empowering them and giving them a set of explicitly declared policies holds true with the core value of respecting the human condition.

Use the Scientific Method

Seek to use models to understand the dynamics of how work is done and how the system of Lean Software Development is operating. Observe and study the system and its capability, and then develop and apply models for predicting its behavior. Collect quantitative data in your studies, and use that data to understand how the system is performing and to predict how it might change when the process is changed.

The Lean Software & Systems community uses statistical methods such as statistical process control charts and spectral analysis histograms of raw data for lead time and velocity to understand system capability. They also use models such as: the Theory of Constraints to understand bottlenecks; The System of Profound Knowledge to understand variation that is internal to the system design versus that which is externally influenced; and an analysis of economic costs in the form of tasks performed to merely coordinate, set up, deliver, or clean up after customer-valued product or services are created. Some other models are coming into use, such as Real Option Theory, which seeks to apply financial option theory from financial risk management to real-world decision making.

The scientific method suggests: we study; we postulate an outcome based on a model; we perturb the system based on that prediction; and we observe again to see if the perturbation produced the results the model predicted. If it doesn't, then we check our data and reconsider whether our model is accurate. Using models to drive process improvements moves it to a scientific activity and elevates it from a superstitious activity based on intuition.

Encourage Leadership

Leadership and management are not the same. Management is the activity of designing processes, creating, modifying, and deleting policy, making strategic and operational decisions, gathering resources, providing finance and facilities, and communicating information about context such as strategy, goals, and desired outcomes. Leadership is about vision, strategy, tactics, courage, innovation, judgment, advocacy, and many more attributes. Leadership can and should come from anyone within an organization. Small acts of leadership from workers will create a cascade of improvements that will deliver the changes needed to create a Lean Software Development process.

Generate Visibility

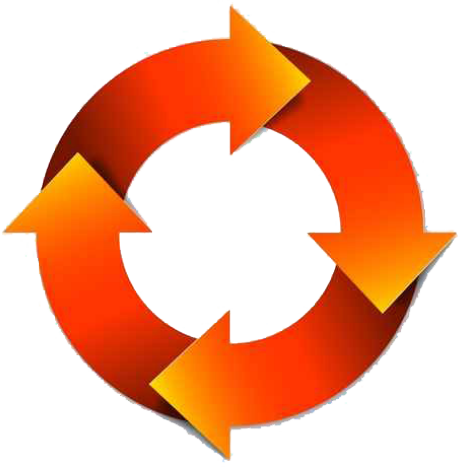
Knowledge work is invisible. If you can't see something, it is (almost) impossible to manage it. It is necessary to generate visibility into the work being undertaken and the flow of that work through a network of individuals, skills, and departments until it is complete. It is necessary to create visibility into the process design by finding ways of visualizing the flow of the process and by making the policies of the process explicit for everyone to see and consider. When all of these things are visible, then the use of the scientific method is possible, and conversations about potential improvements can be collaborative and objective. Collaborative process improvement is almost impossible if work and workflow are invisible and if process policies are not explicit.

Reduce Flow Time

The software development profession and the academics who study software engineering have traditionally focused on measuring time spent working on an activity. The Lean Software Development community has discovered that it might be more useful to measure the actual elapsed calendar time something takes to be processed. This is typically referred to as Cycle Time and is usually qualified by the boundaries of the activities performed. For example, Cycle Time through Analysis to Ready for Deployment would measure the total elapsed time for a work item, such as a user story, to be analyzed, designed, developed, tested in several ways, and queued ready for deployment to a production environment.

Focusing on the time work takes to flow through the process is important in several ways. Longer cycle times have been shown to correlate with a non-linear growth in bug rates. Hence shorter cycle times lead to higher quality. This is counter-intuitive as it seems ridiculous that bugs could be inserted in code while it is queuing and no human is actually touching it. Traditionally, the software engineering profession and academics who study it have ignored this idle time. However, empirical evidence suggests that cycle time is important to initial quality.

Alan Shalloway has also talked about the concept of "induced work." His observation is that a lag in performing a task can lead to that task taking a lot more effort than it may have done. For example, a bug found and fixed immediately may only take 20 minutes to fix, but if that bug is triaged, is queued and then waits for several days or weeks to be fixed, it may involve several or many hours to make the fix. Hence, the cycle time delay has "induced" additional work. As this work is avoidable, in Lean terms, it must be seen as "waste."



The third reason for focusing on cycle time is a business related reason. Every feature, function, or user story has a value. That value may be uncertain but, nevertheless, there is a value. The value may vary over time. The concept of value varying over time can be expressed economically as a market payoff function. When the market payoff function for a work item is understood, even if the function exhibits a spread of values to model uncertainty, it is possible to evaluate a "cost of delay." The cost of delay allows us to put a value on reducing cycle time.

With some work items, the market payoff function does not start until a known date in the future. For example, a feature designed to be used during the 4th of July holiday in the United States has no value prior to that date. Shortening cycle time and being capable of predicting cycle time with some certainty is still useful in such an example. Ideally, we want to start the work so that the feature is delivered "just in time" when it is needed and not significantly prior to the desired date, nor late, as late delivery incurs a cost of delay. Just-in-time delivery ensures that optimal use was made of available resources. Early delivery implies that we might have worked on something else and have, by implication, incurred an opportunity cost of delay.

As a result of these three reasons, Lean Software Development seeks to minimize flow time and to record data that enables predictions about flow time. The objective is to minimize failure demand from bugs, waste from over-burdening due to delay in fixing bugs, and to maximize value delivered by avoiding both cost of delay and opportunity cost of delay.

Reduce Waste to Improve Efficiency

For every valued-added activity, there are setup, cleanup and delivery activities that are necessary but do not add value in their own right. For example, a project iteration that develops an increment of working software requires planning (a setup activity), an environment and perhaps a code branch in version control (collectively known as configuration management and also a setup activity), a release plan and performing the actual release (a delivery activity), a demonstration to the customer (a delivery activity), and perhaps an environment teardown or reconfiguration (a cleanup activity.) In economic terms, the setup, cleanup, and delivery activities are transaction costs on performing the value-added work. These costs (or overheads) are considered waste in Lean.

Any form of communication overhead can be considered waste. Meetings to determine project status and to schedule or assign work to team members would be considered a coordination cost in economic language. All coordination costs are waste in Lean thinking. Lean software development methods seek to eliminate or reduce coordination costs through the use of colocation of team members, short face-to-face meetings such as standups, and visual controls such as card walls.

The third common form of waste in Lean Software Development is failure demand. Failure demand is a burden on the system of software development. Failure demand is typically rework or new forms of work generated as a side-effect of poor quality. The most typical forms of failure demand in software development are bugs, production defects, and customer support activities driven out of a failure to use the software as intended. The percentage of work-in-progress that is failure demand is often referred to as Failure Load. The percentage of value-adding work against failure demand is a measure of the efficiency of the system.

The percentage of value-added work against the total work, including all the non-value adding transaction and coordination costs, determines the level of efficiency. A system with no transaction and coordination costs and no failure load would be considered 100% efficient.

Traditionally, Western management science has taught that efficiency can be improved by increasing the batch size of work. Typically, transaction and coordination costs are fixed or rise only slightly with an increase in batch size. As a result, large batches of work are more efficient. This concept is known as "economy of scale." However, in knowledge work problems, coordination costs tend to rise non-linearly with batch size, while transaction costs can often exhibit a linear growth. As a result, the traditional 20th Century approach to efficiency is not appropriate for knowledge work problems like software development.

It is better to focus on reducing the overheads while keeping batch sizes small in order to improve efficiency. Hence, the Lean way to be efficient is to reduce waste. Lean software development methods focus on fast, cheap, and quick planning methods; low communication overhead; and effective low overhead coordination mechanisms, such as visual controls in kanban systems. They also encourage automated testing and automated deployment to reduce the transaction costs of delivery. Modern tools for minimizing the costs of environment setup and teardown, such as modern version control systems and use of virtualization, also help to improve efficiency of small batches of software development.

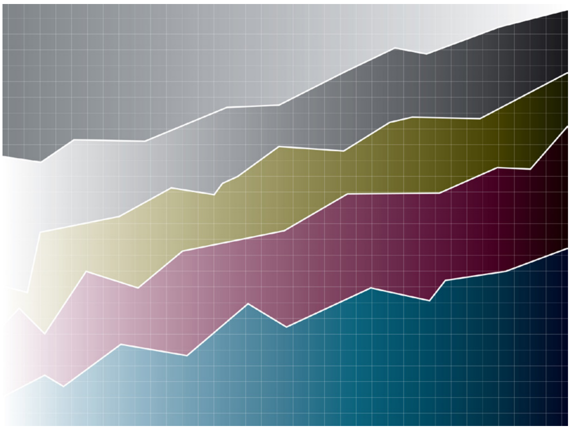
Practices

Lean Software Development does not prescribe practices. It is more important to demonstrate that actual process definitions are aligned with the principles and values. However, a number of practices are being commonly adopted. This section provides a brief overview of some of these.

Cumulative Flow Diagrams

Cumulative Flow Diagrams have been a standard part of reporting in Team Foundation Server since 2005. Cumulative flow diagrams plot an area graph of cumulative work items in each state of a workflow. They are rich in information and can be used to derive the mean cycle time between steps in a process as well as the throughput rate (or "velocity"). Different software development lifecycle processes produce different visual signatures on cumulative flow diagrams. Practitioners can learn to recognize patterns of dysfunction in the process displayed in the area graph. A truly Lean process will show evenly distributed areas of color, smoothly rising at a steady pace. The picture will appear smooth without jagged steps or visible blocks of color.

In their most basic form, cumulative flow diagrams are used to visualize the quantity of work-in-progress at any given step in the work item lifecycle. This can be used to detect bottlenecks and observe the effects of "mura" (variability in flow).



Visual Controls

In addition to the use of cumulative flow diagrams, Lean Software Development teams use physical boards, or projections of electronic visualization systems, to visualize work and observe its flow. Such visualizations help team members observe work-in-progress accumulating and enable them to see bottlenecks and the effects of "mura." Visual controls also enable team members to self-organize to pick work and collaborate together without planning or specific management direction or intervention. These visual controls are often referred to as "card walls" or sometimes (incorrectly) as "kanban boards."

Virtual Kanban Systems

A kanban system is a practice adopted from Lean manufacturing. It uses a system of physical cards to limit the quantity of work-in-progress at any given stage in the workflow. Such work-in-progress limited systems create a "pull" where new work is started only when there are free kanban indicating that new work can be "pulled" into a particular state and work can progress on it.

In Lean Software Development, the kanban are virtual and often tracked by setting a maximum number for a given step in the workflow of a work item type. In some implementations, electronic systems keep track of the virtual kanban and provide a signal when new work can be started. The signal can be visual or in the form of an alert such as an email.

Virtual kanban systems are often combined with visual controls to provide a visual virtual kanban system representing the workflow of one or several work item types. Such systems are often referred to as "kanban boards" or "electronic kanban systems."

Small Batch Sizes / Single-Piece Flow

Lean Software Development requires that work is either undertaken in small batches, often referred to as "iterations" or "increments," or that work items flow independently, referred to as "single-piece flow." Single-piece flow requires a sophisticated configuration management strategy to enable completed work to be delivered while incomplete work is not released accidentally. This is typically achieved using branching strategies in the version control system. A small batch of work would typically be considered a batch that can be undertaken by a small team of 8 people or less in under 2 weeks.

Small batches and single-piece flow require frequent interaction with business owners to replenish the backlog or queue or work. They also require a capability to release frequently. To enable frequent interaction with business people and frequent delivery, it is necessary to shrink the transaction and coordination costs of both activities. A common way to achieve this is the use of automation.

Automation

Lean Software Development expects a high level of automation to economically enable single-piece flow and to encourage high quality and the reduction of failure demand. The use of automated testing, automated deployment, and software factories to automate the deployment of design patterns and creation of repetitive low variability sections of source code will all be commonplace in Lean Software Development processes.

Kaizen Events

In Lean literature, the term kaizen means "continuous improvement" and a kaizen event is the act of making a change to a process or tool that hopefully results in an improvement.

Lean Software Development processes use several different activities to generate kaizen events. These are listed here. Each of these activities is designed to stimulate a conversation about problems that adversely affect capability and, consequently, ability to deliver against demand. The essence of kaizen in knowledge work is that we must provoke conversations about problems across groups of people from different teams and with different skills.

Daily Standup Meetings

Teams of software developers, often up to 50, typically meet in front of a visual control system such as a whiteboard displaying a visualization of their work-in-progress. They discuss the dynamics of flow and factors affecting the flow of work. Particular focus is made to externally blocked work and work delayed due to bugs. Problems with the process often become evident over a series of standup meetings. The result is that a smaller group may remain after the meeting to discuss the problem and propose a solution or process change. A kaizen event will follow. These spontaneous meetings are often referred to as spontaneous quality circles in older literature. Such spontaneous meetings are at the heart of a truly kaizen culture. Managers will encourage the emergence of kaizen events after daily standup meetings in order to drive adoption of Lean within their organization.

Retrospectives

Project teams may schedule regular meetings to reflect on recent performance. These are often done after specific project deliverables are complete or after time-boxed increments of development known as iterations or sprints in Agile software development.

Retrospectives typically use an anecdotal approach to reflection by asking questions like “what went well?”, “what would we do differently?”, and “what should we stop doing?”

Retrospectives typically produce a backlog of suggestions for kaizen events. The team may then prioritize some of these for implementation.

Operations Reviews

An operations review is typically larger than a retrospective and includes representatives from a whole value stream. It is common for as many as 12 departments to present objective, quantitative data that show the demand they received and reflect their capability to deliver against the demand. Operations reviews are typically held monthly. The key differences between an operations review and a retrospective is that operations reviews span a wider set of functions, typically span a portfolio of projects and other initiatives, and use objective, quantitative data. Retrospectives, in comparison, tend to be scoped to a single project; involve just a few teams such as analysis, development, and test; and are generally anecdotal in nature.

An operations review will provoke discussions about the dynamics affecting performance between teams. Perhaps one team generates failure demand that is processed by another team? Perhaps that failure demand is disruptive and causes the second team to miss their commitments and fail to deliver against expectations? An operations review provides an opportunity to discuss such issues and propose changes. Operations reviews typically produce a small backlog of potential kaizen events that can be prioritized and scheduled for future implementation.

There is no such thing as a single Lean Software Development process. A process could be said to be Lean if it is clearly aligned with the values and principles of Lean Software Development. Lean Software Development does not prescribe any practices, but some activities have become common. Lean organizations seek to encourage kaizen through visualization of workflow and work-in-progress and through an understanding of the dynamics of flow and the factors (such as bottlenecks, non-instant availability, variability, and waste) that affect it. Process improvements are suggested and justified as ways to reduce sources of variability, eliminate waste, improve flow, or improve value delivery or risk management. As such, Lean Software Development processes will always be evolving and uniquely tailored to the organization within which they evolve. It will not be natural to simply copy a process definition from one organization to another and expect it to work in a different context. It will also be unlikely that returning to an organization after a few weeks or months to find the process in use to be the same as was observed earlier. It will always be evolving.

The organization using a Lean software development process could be said to be Lean if it exhibited only small amounts of waste in all three forms ("mura," "muri," and "muda") and could be shown to be optimizing the delivery of value through effective management of risk. The pursuit of perfection in Lean is always a journey. There is no destination. True Lean organizations are always seeking further improvement.

Lean Software Development is still an emerging field, and we can expect it to continue to evolve over the next decade.

About David J. Anderson

David J. Anderson is a thought leader in managing effective technology development. He leads an international management training and consulting firm, David J. Anderson & Associates Inc. (www.djaa.com), that helps businesses improve their performance through better management policies and decision making.

He has 30 years experience in the high technology industry. He has led software teams delivering superior productivity and quality using innovative agile methods at large companies such as Sprint, Motorola, and Microsoft.

David is the author of three books, **Agile Management for Software Engineering – Applying the Theory of Constraints for Business Results, Kanban – Successful Evolutionary Change for your Technology Business**, and **Lessons in Agile Management: On the Road to Kanban**. David is CEO of Lean-Kanban University, a business dedicated to assuring quality of training in Lean and Kanban throughout the world.

About Lean-Kanban University

Lean-Kanban University (LKU) works to assure the highest quality coaching and training on Kanban for knowledge work and service work worldwide. LKU Accredited Kanban Training™ program partners and Kanban Coaching Professionals™ follow the Kanban Method for evolutionary organizational change.

Lean-Kanban University offers accreditation for Kanban trainers, a professional designation for Kanban coaches, and certification and LKU membership for Kanban practitioners.

www.leankanbanuniversity.com

References

1. Womack, James P., Daniel T. Jones and Daniel Roos, The Machine That Changed the World: The Story of Lean Production, 2007 updated edition, Free Press, 2007
2. Womack, James P., and Daniel T. Jones, Lean Thinking: Banish Waste and Create Wealth in your Corporation, 2nd Edition, Free Press, 2003
3. Beck, Kent et al, The Manifesto for Agile Software Development, 2001 http://www.agilemanifesto.org/
4. Highsmith, James A., Agile Software Development Ecosystems, Addison Wesley, 2002
5. Poppendieck, Mary and Tom Poppendieck, Lean Software Development: An Agile Toolkit, Addison Wesely, 2003
6. Poppendieck, Mary and Tom Poppendieck, Implementing Lean Software Development: From Concept to Cash, Addison Wesley, 2006
7. Poppendieck, Mary and Tom Poppendieck, Leading Lean Software Development: Results are not the Point, Addison Wesley, 2009
8. Reinertsen, Donald G., Managing the Design Factory, Free Press, 1997
9. Reinertsen, Donald G., The Principles of Product Development Flow: Second Generation Lean Product Development, Celeritas Publishing, 2009
10. Sutton, James and Peter Middleton, Lean Software Strategies: Proven Techniques for Managers and Developers, Productivity Press, 2005
11. Anderson, David J., Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results, Prentice Hall PTR, 2003
12. Shalloway, Alan, and Guy Beaver and James R. Trott, Lean-Agile Software Development: Achieving Enterprise Agility, Addison Wesley, 2009
13. Larman, Craig and Bas Vodde, Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-scale Scrum, Addison Wesley Professional, 2008
14. Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum, Addison Wesley Professional, 2010
15. Coplien, James O. and Gertrud Bjornvig, Lean Architecture: for Agile Software Development, Wiley, 2010
16. http://leansystemssociety.org/
17. http://www.leanssc.org/2011/05/leanssc-vision-values-and-mission-1-0beta/
18. Anderson, David J., Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results, Prentice Hall PTR, 2003
19. Anderson, David J., Kanban: Successful Evolutionary Change for your Technology Business, Blue Hole Press, 2010
20. Anderson, David J., Lessons in Agile Management: On the Road to Kanban, Blue Hole Press, 2012

LKU | Lean-Kanban University

www.leankanbanuniversity.com