Lean from the Trenches
Managing Large-Scale Projects with Kanban

Henrik Kniberg; The Pragmatic Programmers, LLC, 2011, 157 pages
ISBN 978-1-934356-85-2

Book report
Jim McDonough, April 27, 2012

Review

This book is primarily a case study of how Lean and Kanban were utilized on a software project for implementing a new digital investigation system for the Swedish National Police Authority. Part I, How We Work, comprises two thirds of the book and addresses this case study. This is followed by Part II, A Closer Look at the Techniques, which describes more of the academic aspects of Agile Software Development processes used in the case study. It includes a foreword by Kent Beck, who states:

> "I've been fortunate enough to work with Henrik a bit, and he told me he really has only one trick: make all the important information visible in one place and then decide what to do together." (p. ix)

These ideas of 'make information visible' and 'decide together' are reinforced throughout the book as the author describes in detail how the team approached the project, freely admitting not adhering purely to any prescribed process with this candid disclaimer:

> "I don't claim that our way of working is perfectly Lean. Lean is a direction, not a place. It's all about continuous improvement." (p. xii)

1 – About the Project

To set the stage for the remainder of the book, the author starts with some background on the case study, including the type of work performed by the customer, the expected time line for implementing the new software releases, the number of participants, the division of work and how the customer was involved with the software development effort. Reinforcing the concept of continuous improvement, the author states that a Lean approach was used not only to facilitate the software design and development, but also to improve the process by which to deliver that software:

> "[The project] is part of a cultural change within [the Swedish National Police Authority], a nationwide Lean initiative throughout the whole organization. So, it made sense to start applying Lean principles to the development process itself too!" (p. 5)

Every project has its own adjustments to be made when applying theory to practice:

> "My main focus was on putting Lean and Agile principles into practice and helping the teams evolve just the right process for their context.
>
> "One of the most important characteristics of a Lean process is that it *keeps evolving*. Sometimes we find better solutions. Sometimes a seemingly great solution yesterday causes a new problem today. Sometimes our environment and circumstances change, forcing us to adapt." (pp. 5-6)

2 – Structuring the Teams

The teams for this project consisted of three feature teams, one requirements team and one testing team, though there was some overlap between the members of each team as both the requirements team and testing team had representatives in each of the three feature teams. Also, the teams all were located on the same floor of the same building. Both the collocation and team member overlap facilitated communication amongst all participants. The author states that this was a beneficial arrangement when the project scaled up:

> “In the past, the teams were organized by specialty. We had a distinct requirements team, a distinct test team, and distinct development teams that did not have embedded testers or analysts. That didn't scale very well, because as more people were added to the project, communication problems developed. Teams tended to communicate with other teams through documents rather than talking, and they tended to blame problems on each other. Teams also tended to focus on getting their part of the work done instead of the whole product.” (pp. 10-11)

This arrangement, the author adds, was not in place at the start, but became the team structure after gradually evolving.

3 – Attending the Daily Cocktail Party

A glimpse is provided into the use of a technique to which many refer as the daily stand-up scrum, short meetings, maximum of fifteen minutes, where the status and progress of the team is discussed amongst its members. There were three tiers of these meetings taking place on this project:

- First Tier – The three feature teams; one starting at 9:15am and the other two starting at 9:30am.
- Second Tier – Three synchronization meetings per specialty: one for the requirements team, one for the testing team and one for development synchronization of the three first tier feature teams, with all meetings taking place concurrently starting at 9:45am.
- Third Tier – Project synchronization of the three second tier meetings, starting at 10:00am.

This facilitated effective communication:

> “That's it. A total of seven stand-up meetings every day, organized into three layers. Each meeting is *timeboxed* to fifteen minutes, each meeting has a core set of participants who show every day, and each meeting is public, so anybody can visit any meeting if they want to learn what is going on or have something to contribute. And it's all over by 10:15am.
>
> “Many problems that would otherwise result in the creation of documents and process rules are resolved directly at these morning meetings. One concrete example is deciding which team is to develop which feature; another example is deciding whether to spend our time development customer-facing functionality or customer-invisible improvements to the technical infrastructure. Instead of setting up policy rules for this, the teams simply talk about this during the daily meetings and make decisions on-the-fly based on the current situation. This is the key to staying agile in a big project and not getting bogged down in bureaucracy.” (p. 17)

4 – The Project Board

Referring to it as the communication hub of the project, the author describes the characteristics of the project board,

> “Ours is a several-meter-long whiteboard showing all key project features flowing through the pipeline from requirements, development and system test, all the way into production.” (p. 19)

complete with photographs of the actual board used during this project,

> “If you are into Kanban, you'll recognize this as a *Kanban system*, which means that we track the flow of value from idea to production and that we limit the amount of work in progress at each step of the process.” (p. 19)

and uses a traffic analogy to describe how work flows through the system.

> “Just as with a traffic system, a blocker that stays around for too long will cause ripple effects throughout the whole system. Plus, nothing will flow faster than the bottleneck section of the road, so we focus all efforts on resolving these bottlenecks.” (p. 25)

5 – Scaling the Kanban Boards

The author explains how the project board enabled managing the challenge of increasing the size of the staff as the project progressed,

> “The speed of a project is largely determined by how well everyone understands what's going on. If everyone knows where we are right now and where we're going, it's much easier for everyone to move in the same direction.
>
> “That's why we created the project board. It's a way to keep track of the big picture by showing project features as they move from requirements to development, to system test, and into production.
>
> “This board had a strong effect on the culture of the organization. Now we could *see*! And we all had the *same picture*!” (p. 27)

but the project board was not the only Kanban board used on the project:

> “So, the project board contains feature cards, and each feature team has their own board with the features they are working on plus the associated task breakdown. Imagine that you “double-click” a feature on the project board and “zoom in” to the corresponding team board to see which tasks are involved in that feature and who is working on what task.” (p. 29)

Since this project was not following a prescribed formula, but instead team members were discovering what worked for them, there was no attempt to have each team board conform to some arbitrary format:

> “As you can see in the pictures, each team has their own board layout. We have not tried to standardize this. Instead, we let each team find whatever board structure works best for them.” (p. 30)

6 – Tracking the High-Level Goal

The high level goal was posted on the Kanban board for all to see. This alleviated the problem experienced on some projects where it can only be assumed that everyone is aware of the goal, when in reality there may be wide range of opinions on what constitutes the high level goal.

Also, the goal was reviewed periodically in the project synchronization meeting to determine whether or not the participants still felt the goal was achievable. A simple method of voting by holding up fingers, from 1 to 5, would determine whether there was any reason to adjust the high level goal.

> “I can strongly recommend having a clear goal and periodic reality check, no matter what type of project you are involved in. The cost of doing this is very low, and the benefit is very high!” (p. 34)

7 – Defining Ready and Done

The definition of "done" appeared at the top of most columns on the project board, and also served to indicate the definition of "ready" for the next column, the two most important being the "ready for development" and "ready for system test".

> "These two policy statements – *definition of ready for development* and *definition of ready for system test* – have significantly improved collaboration between the teams. This improvement stood out clearly when I did a short survey to check what people thought about tall the process changes so far." (p. 38)

8 – Handling Tech Stories

The author explains what tech stories are and how important it is to incorporate them into the flow of work:

> "Tech stories are things that need to get done but that are uninteresting to the customer, such as upgrading a database, cleaning out unused code, refactoring a messy design, or catching up on test automation for old features. We call these internal improvements, but in retrospect, tech stories is probably a better term, since we're talking about stuff to be done with the product, not process improvements." (p. 39)

The author cites a humorous story about a measurement unit arising from this project for gauging the effort involved in implementing tech stories:

> "One of the classes in our code base was getting way out of control and needed some significant refactoring, but there was some resistance to spending time on that. So, one of the team leads printed out the whole class and laid it across the conference table! It was more than 7 meters long (23 feet)!
>
> "Looking at that monstrous printout, everyone clearly saw that we needed a tech story to fix that class immediately! No argument needed. This also illustrated the consequence of being in a hurry an not paying enough attention to design.
>
> "We had some fun speculating about future developments along this theme. How about if we estimate features in code-meters and measure velocity in code-meters per day? We could even separate *ideal* code-meters (how long the code would be if we kept it really clean), with *actual* code-meters. Subtract those two, and you get technical debt – in meters! We could even draw a line in the floor to symbolize how much technical debt we have ('Hey look, we have 23 meters of debt!')." (pp. 42-43)

9 – Handling Bugs

The author recommends both continuous system testing and fixing bugs immediately, as opposed to saving up batches of completed work to be system tested all at once. A convincing argument is presented for how the continuous system testing will save time and effort in the long run.

Also suggested is a practical policy toward bugs, one in which not all bugs will be fixed, as there are some bugs of such low significance that their chances of being addressed are nil. Better to acknowledge publicly that these bugs will not be addressed rather than providing false hope they will be fixed.

Some bugs are so important they get their own tracking sticker on the project board. Others that are of less importance will be tracked in a bug queue. The size of the bug queue is limited. When a new bug arises, and is not important enough to get its own tracking sticker on the project board, a decision is made whether it should be added to the full bug queue, based on whether it is more important than some other bug already in the bug queue. If not, the new bug is ignored; otherwise it replaces on the queue the bug to which it is more important. This is how the number of bugs is managed to only those which will be corrected.

> “Limiting the size of the bug tracker builds trust. The bug list is short, but stuff in there actually does get fixed, rather than sitting around for months without anything happening. If a bug is unlikely to be fixed … we are honest about that from [the] start, instead of building up false expectations.” (p. 49)

10 – Continuously Improving the Process

Improving the software development process itself consisted of holding team retrospectives, engaging the staff in process improvement workshops and managing the rate of change. This is one area which seems to get very little attention on those software development projects where management believes merely following a documented process is sufficient. Indeed, as the author explains:

> “Our process was by no means designed up front. I would never have been able to figure all this out, especially not alone. And even if I could, there would be no buy-in from the project participants, so any process designed up front would never have reached further than the document it was written on.
>
> “Our process was discovered rather than designed.” (p. 53)

Despite having a process facilitating change, it was necessary to throttle the rate of change to avoid overwhelming the staff with too many changes at once:

> “But the rate of change also caused confusion, especially for the majority of the people who weren't in the cross-team and who saw lots of change happening without always being given a chance to understand or discuss the change.” (p. 54)

This not only limits concurrent changes to a manageable level but also endorses the proposition that those who are subject to changes should be able to contribute to the change process.

11 – Managing Work in Progress

The project board has columns for work in progress (WIP) as well as buffering columns used as staging areas to manage work moving between work in progress columns. The author uses a mailbox analogy to explain the relationship between work in progress and their buffers – a mailbox can hold only so much mail before it needs to be cleared of content – and presents a good argument for limiting the amount of work which can be active for a given work in progress column.

> “The purpose of WIP limits is to avoid too much multitasking and overloading a downstream process. If the testers have too much work to do, we don't want developers to keep building new features and adding to their workload – instead, they should focus on helping test. WIP limits act as an alert signal to highlight the problem before it gets out of hand.” (p. 69)

12 – Capturing and Using Process Metrics

On this project there were two metrics being tracked:

- Features per week (velocity)
- Weeks per feature (cycle time)

The velocity metric was used in two ways: 1) as a reality check to make sure the release plan was realistic, and 2) to predict approximately how many features will be done by a certain date.

The cycle time metric was used to predict how long it would take for a feature to be completed.

The author provides reasons for not using some of the other tracking techniques commonplace on software development projects, such as using story points, measuring cycle time all the way to production and cumulative flow.

13 – Planning the Sprint Release

A planning meeting is convened to determine what to do next. The resulting list of features goes onto the project board in the column titled "Next Ten Features". The author admits to some divergence from Scrum procedures by stating that the team does not commit to a specific set of features for the next sprint, as in Scrum, nor do they even use sprints. They simply agree on what is next to be done as the velocity has not be stable enough to enable predicting how many features could get done in the short term. Features are chosen based upon the following criteria (p. 86):

Business value –	which features will the customer be happiest to see?
Knowledge –	which features will generate knowledge (and thereby reduce risk)?
Resource utilization –	we want a balance of feature areas so that each team has stuff to work on.
Dependencies –	which features are best built together?
Testability –	which features are most logical to test together and should therefore be developed in close proximity to each other?

14 – How We Do Version Control

Software management problems began to ensue upon scaling from thirty to sixty people. At the time the version control system being used was unstable, containing broken trunks and branches for long periods. To rectify this, a mainline model was instituted consisting of a stable trunk with branches for each team and a branch for system test.

> "The version control system is really the heart of any multiteam development project. As the organization becomes more Lean and Agile, the version control system usually needs to be evolved as well. So, keep an eye on this. Find out how long it takes to change one single line of code and get it into production. That may well be the most important metric in the project!" (p. 94)

15 – Why We Use Only Physical Kanban Boards

With all the software tools available for managing projects, it was refreshing to learn that this project had used a physical project board instead of an electronic counterpart. The author explains that the physical version was much more adaptable to the evolution of the project board, with the changes required being easily applied to a physical entity very easily by just about anyone, whereas an electronic version would require expert knowledge of the tool itself, and still could not keep up with the rate of change as easily as a physical version.

Another reason mentioned for using a physical board was collaboration between the participants. The daily cocktail parties (stand-up scrums) took place in front of the physical boards, providing a platform to facilitate the interaction for the discussion in progress.

> "One clear pattern I've noticed with all my clients is that boards like this can change the culture of an organization, and this definitely happened [on this] project. I could see how the patterns of interaction changed and how trust between the teams improved by collaborating in front of the boards every day. During our first project-level retrospective, one of the the first items that came up under *keep doing* was 'keep using Kanban boards to visualize the work.'" (p. 97)

16 – What We Learned

Based on his experience on this project, the author shares his observation on what makes a great process:

> "A great process isn't designed; its *evolved*. So, the important thing isn't your process; the important thing is your process for *improving* your process." (p. 99)

A summary of some of the things learned on this project are:

- Know Your Goal
- Experiment
- Embrace Failure
- Solve Real Problems
- Have Dedicates Change Agents
- Involve People

17 – Agile and Lean in Nutshell

The author provides brief "... in a Nutshell" summaries of the following software development concepts:

- Agile
- Lean
- Scrum
- XP
- Kanban

Agile is described as starting in 2001 when seventeen software development leaders met at a ski resort in Utah, resulting in the creation of the Agile Manifesto, though many of the techniques encompassing Agile already had been developed prior to this date.

Lean is described as originating from Japanese manufacturing techniques, specifically the Toyota Production System, with those principles as adapted to software outlined by Mary and Tom Poppendieck:

- Optimize the Whole
- Eliminate Waste
- Build Quality In
- Learn Constantly
- Deliver Fast
- Engage Everyone
- Keep Getting Better

Scrum is described as an approach developed in the early 1990s by Jeff Sutherland and Ken Schwaber, based on principles of empirical process control and complex adaptive systems theory, with explanations for scrum terms such as product backlog, sprints, cross-functional teams and continuous process adjustment.

XP is described as an approach developed in the mid-1990s by Kent Beck, based on values of simplicity, communication, feedback, courage and respect. It is shown how XP and Scrum complement each other and provides further detail for XP terms such as continuous integration, pair programming, test-driven development, collective ownership of code and incremental design improvement.

Kanban, a Japanese word meaning "visual card", is described as a technique associated with Lean, with further definition presented for its characteristics of visualizing workflow, limiting work in progress and measuring and managing cycle time. As these conceptual aspects of Kanban are revealed, along with a series of Kanban images depicting an example time progression, it serves to clarify the underlying principles at work for the project board which had been used as the basis for the case study.

18 – Reducing the Test Automation Backlog

After stating what seems to be a barrier to progress for many software development organizations – the absence of automated tests for legacy code – the author presents options for addressing this difficulty:

1. Ignore the problem
2. Rebuild the system from scratch using test-driven development
3. Start a separate test automation project
4. Let the team improve test coverage with each iteration

Upon identifying option 4 as the most practical based on his experience, the author presents a road map of how to approach the daunting task of providing new automated tests for existing software.

> "Regardless of the test automation backlog, each *new* feature should include an automated test at the feature level. That's the XP practice known as *customer acceptance tests*. Not doing this is what got your system into this mess in the first place.
>
> "But in addition to implementing new stories, we want to spend some time automating old test cases." (p. 122)

It is commendable that the author addresses the problem of test automation backlog, as test automation easily can be perceived as a useful tool applying only to new development.

For those who have yet to embrace the concept of automated testing, the author makes a convincing case for its use:

> "Note that *manual test cost* is incurred every time the test is run, while *automation cost* is incurred only once. So, time spent writing test automation code is actually an investment, not a cost." (p. 120)

19 – Sizing the Backlog with Planning Poker

Planning Poker, a term coined by James Grenning and popularized by Mike Cohn, is a technique used for estimating the effort involved in software development tasks. The author shows how to "play" the estimating game using playing cards which have five different symbols to signify the estimated effort:

S – Small effort
M – Medium effort

L – Large effort
? – Don't know
Coffee cup – Time for a break

Each team member gets one of each card. A feature is presented for estimation. Each team member presents one of the cards, face down. When all cards have been played, each one is turned over to reveal its symbol. Virtual consensus on estimated effort is achieved when there is a convergence to one type of card. In cases of wide divergence, a discussion ensues to determine why there is such disagreement on estimated effort, and the game continues into another round, repeating the cycle until there is some agreement on estimated effort.

This use of cards eliminates the influence some team members would have upon others in cases where each person is asked in series to verbally state their estimate, or where some team members, due to confusion or inattention, simply specify an estimate of effort already stated by someone else.

20 – Cause-Effect Diagrams

A thorough treatment of the use of cause and effect diagrams shows how these diagrams can highlight problems in the process rather than just the symptoms. One technique presented is the A3 format typical of Lean, while another is the use of the Five Whys.

The author shows how to use boxes to diagram aspects of a process. Connecting arrows show the relationships between the boxes. Those boxes with only inbound arrows represent problems; those with only outbound arrows represent root causes; those with both inbound and outbound arrows represent symptoms. Once identified, the root causes can be addressed.

An interesting result from the use of this type of diagram is the emergence of a series of symptom boxes related to each other via a circular set of connecting arrows, known as both vicious cycles and reinforcing loops.

> "Recurring problems almost always involve [reinforcing] loops … but they may take some time to find. Spotting these will greatly increase your likelihood of solving the problem effectively and permanently!" (p. 136)

A cause-effect diagram is created by following a simple sequence of steps:

> "Here's the basic process:
>
> 1. Select a problem – anything that's bothering you – and write it down.
> 2. Trace 'upward' to figure out the business consequences, the 'visible damage' that your problem is causing'.
> 3. Trace 'downward' to find the root cause (or causes).
> 4. Identify and highlight vicious cycles (circular paths).
> 5. Iterate these steps a few times to refine and clarify your diagram.
> 6. Decide which root causes to address and how (that is, which countermeasures to implement)." (p. 134)

The use of cause-effect diagrams is summarized as helping teams to

- Create a common understanding
- Identify how problems affect the business
- Find root causes
- Identify and eliminate vicious cycles

Summary

This book contains a good balance of Agile theory and its practical application. The book is short and easy to read and the author does a superb job of describing how Agile, Lean and Kanban were used in practice on a specific project as well as presenting general information about these concepts. Though no book describing an actual implementation of the use of Agile practices should be used as a model for others to copy, this book presents the case study in a way where the reader can appreciate *how information is made available*, enabling the team to *decide together* how to proceed.

Notable passages

> “One person acted as the main “customer” (or “buyer”) to the project. She had a list of features at a pretty high level. We called them feature areas, which roughly equates to what agile folks would call epics.” (p. 7)
>
> “To the casual observer glancing at the [project] board, this system might look like a waterfall process: requirements analysis → development → system test → acceptance test → production. There's a big difference, though. In a waterfall model, the requirements are all completed before development starts, and development is completed before testing starts. In a Kanban system, these phases are all going on in parallel. While one set of features is begin acceptance-tested by users, another set of features is being system tested, a third set of features is begin developed and a fourth set of features is being analyzed and broken into user stories. It's a continuous flow of value from idea to production.” (p. 21)
>
> “Using Kanban to Discover Scrum – This seems to be a general pattern: I see many Kanban teams that gradually discover (or sometimes rediscover) the value of many of the Scrum practices. In fact, sometimes Kanban teams start doing Kanban because they didn't like Scrum and then later discover that Scrum was actually pretty good and their problems had been exposed by Scrum, not caused by it. Their real problem was that they had been doing Scrum too much 'by the book' instead of inspecting and adapting it to their context.” (p. 22)
>
> “If you do cause-effect analysis on you bugs, you'll find that bugs aren't really a problem; they are a symptom. Bugs in your product are a symptom of bugs in your *process*. If you focus on fixing your process, you'll dramatically reduce the number of new bugs in your product. It's just like if you focus on fire prevention, you'll reduce the need for fire fighting.” (p. 52)
>
> “The stated purpose of [the process improvement workshop] is to clarify and improve our way of working. One of my most important tasks as coach was to set up and facilitate these process improvement workshops until they became part of the culture.” (p. 55)
>
> “Each weekly process improvement workshop caused a flurry of change, mostly changes for the better. However, after a while, we realized that we were changing too much, too fast. This was an interesting problem. In most organizations I've worked with, the problem is that there is too *little* process change – everyone is stuck in their current ineffective process. Here we had reached the opposite problem. We were making *lots* of changes, and it takes days or sometimes weeks for a significant process change to ripple through a 60-person project. Many team members got confused (and sometimes frustrated) when the cross-team introduced new changes before the dust from previous changes had settled.” (p. 62)
>
> “...Kanban doesn't provide many specific rules. It is up to you to decide things like how many boards to use on a project That's the beauty (and pain) of Kanban – it's flexible, and you figure things out as you go along. Just stick to this rule of thumb: “When in doubt, chose the simplest solution.'” (p. 64)
>
> “If we had an electronic Kanban board, we could use a projector to display it on the wall. But we would lose most of the interaction, the part where people pick up a card from the wall and wave it

while talking about it, or where people write stuff on the cards or move them around during the meeting. People would most likely update the board while sitting at their desk – which is more convenient but less collaborative.” (p. 97)

“Most people like change; they just don't like to *be changed*. So, don't make any change without first involving the people who will be affected by it. Forcing people to change is usually ineffective, unnecessary, and, well, cruel. If people resist your great change proposal, you probably haven't made the problem clear enough. Or you're solving the wrong problem. Go back to your cause-effect diagram … and think again!” (p. 100)

“Agile and Lean can be seen as cousins with common values but different origins. Lean arose from manufacturing. Agile arose from software development. Both sets of principles fit well together and are very broadly applicable. More and more software development organizations are discovering how to combing these principles to cover the whole chain from product concept to delivery.” (p. 106)

“Most teams like Planning Poker because it takes much of the pain out of estimating and instead turns it into a simple and fun process. The greatest value is really in the conversations that get triggered while playing; the estimate itself is just a positive side effect.” (p. 129)

“[An] organization was doing Scrum but was having some problems. The cause-effect diagram that emerged after interviews and workshops showed that they weren't doing Scrum correctly, and this was causing the problems.” (p. 144)