WHY KANBAN?

An Introduction to Kanban and Lean Software Development

Troy Tuttle, Project Lead Consultant with AdventureTech Group

Troy.Tuttle@adventuretechgroup.com
TroyLTuttle@gmail.com

blog.troytuttle.com
twitter.com/troytuttle
Motivation

What is my motivation?

- A practicing “Pragmatic Agilist”
  - Context is King
- Helping developers with project success by identifying and avoiding project dysfunction
- Defining family-friendly definitions of successful projects.
What is Kanban?

Is Kanban the next martial arts fad?
Is Kanban the latest miracle cure?
What is Kanban?

Is Kanban a …

… silver bullet?
Kanban is a transparent, work-limited, pull system. [2]
Kanban is the result of practitioners applying lean principles to software engineering.
Kanban is influenced heavily by existing first-generation Agile methodologies.
Kanban is more like a tool than a methodology.
Kanban Focus for the Agile Folks

Rather than focusing on being Agile which *may* (and *should*) lead to being successful, **Kanban focuses on becoming successful, which *may lead to being Agile.*
The Pillars of Lean

- Pull
- Continuous Flow
- Customer Value
- Waste Elimination
- Continuous Improvement

Founder of Toyota Production Systems
Taiichi Ohno

Quality Guru:
System of Profound Knowledge
W. Edwards Deming
Kanban Is

Kanban is translated as “Visual Card”
Kanban for Software Principles

- Pull value through the Value Stream
- Limit WIP (work in progress)
- Make it visible!
Lean software engineering starts with mapping the value stream.

From the lips of the stakeholder to a production system.

Try mapping your own value stream.

(Great Reading on the subject of software value streams from Mary and Tom Poppendieck)
Kanban’s Lean Heritage

- Don’t **build features** that nobody needs **right now**
- Don’t **write** more **specs** than you can **code**
- Don’t **write** more code than you can **test**
- Don’t **test** more code than you can **deploy**
Kanban Pull
Kanban Pull

- Phase-based development (waterfall) transfers the entire batch from state to state.
- Existing Agile development transfers small batches from state to state (iterations).
- Lean development’s goal is to transfer one piece at a time (*one piece flow*).

*Think about watering your garden with a bucket versus a hose.* [1]
Limiting Work In Progress

- Given a value stream, a WIP limit governs the number of work items that can be in a given state at any instant.
- WIP limits are designed to reduce multi-tasking, maximize throughput, and enhance teamwork.
- If there is a problem, we “stop the line” instead of starting work on a new item.
## Limiting Work In Progress

### Multitasking Exercise!

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>II</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>III</td>
</tr>
</tbody>
</table>
20% time is lost to context switching per task, so fewer tasks means less time lost *(from Gerald Weinberg, Quality Software Management: Systems Thinking)*
Limiting Work In Progress

Performing tasks sequentially yields results sooner.

(Watch service organizations to see this principle in action.)
Limiting Work In Progress

New work items can only be pulled into a state if there is capacity under the WIP limit.
Little’s Law for Queuing Theory

Total Cycle Time = Number of Things in Progress / Average Completion Rate

- The only way to reduce cycle time is by either reducing the WIP, or improving the average completion rate.
  - Achieving both is desirable.
  - Limiting WIP is easier to implement by comparison.
Visualize!

Put work items on a white board and make it visible!

Then have the conversation with the stakeholder, “this is what we have in process now, to take on new work, what shall we back out of the system?”
“A minimal marketable feature is a chunk of functionality that delivers a subset of the customer’s requirements, and that is capable of returning value to the customer when released as an independent entity”

-- M Denne & H Cleland-Huang, Software by Numbers
MMF’s can be very small (10 minute bug fix), or they can be very large (weeks of feature development). It depends on the context and what the business determines as valuable.

Larger MMF’s are decomposed into granular features. These should be as small as possible and still be independently testable. Traditional XP-style user stories work well.

This relationship can also be expressed as:

\[ \text{feature sets} \rightarrow \text{features}. \]

For Agile practitioners, MMF’s in Kanban often replace the practice of time-boxed iterations.
A two-tiered Kanban board with MMF’s and user stories.

<table>
<thead>
<tr>
<th>Queue (4)</th>
<th>MMF (3)</th>
<th>Proposed</th>
<th>WIP(5)</th>
<th>Done</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example of MMF > User Story > Scenario:

- The system should allow company users to resolve duplicate customer records.
  - As a data processor, I need to merge two or more duplicate records into one.
    - Before merging records, confirm user has “merge” role permissions in system.
    - When merging customer records, verify at least one record contains valid demographic information.
    - After successful merge, log merge relationship information to merge log table.
Contrary to popular belief, Kanban does not require dropping of estimations.

However, many Kanban teams find themselves estimating far less frequently than first generation Agile teams or classic phased-based development teams.
Kanban practitioners view detailed estimates through the value stream prism.

- Detailed estimates are historically inaccurate in software development.
- Steve McConnell in *Software Estimation – Demystifying the Black Art* taught us that highly *precise* estimates are not usually highly *accurate*.

- The process of estimation does not get the code base any closer to production—it can be viewed as waste in Lean terms.
The Estimation Game

- In Kanban, detailed estimation can be replaced with cycle time metrics and analysis.
  - Cycle time is the elapsed time between starting a work item and finishing it.
- If MMF’s vary in size, then use relative estimating to determine grouping.
  - Use T-Shirt sizing S, M, L
  - Small = 3 days, Medium = 8 days, Large = 20 days
- With known cycle times, Service Level Agreements can be established with stakeholders:
  - “Once we start work on a medium MMF, we expect to complete it in 8 days.”
Lead Time

- Lead time is customer facing.
  - Lead time starts when the request is made and stops when the MMF is delivered.

- Lead time measures cycle times plus the backlog size.
  - Large backlogs mean longer lead times
  - Backlogs are “inventory”
  - In Lean, inventory is viewed as waste
The Estimation Game

- Kanban allows a team to stop playing the estimation game where value is placed on predicting when software will be completed.
- Instead, teams can project software completion rates from actual data (cycle and lead times).
- The effort that formerly went into improving prediction practices can now go to improving actual software delivery.
- Estimate when the “cost of delay” is high.
Kanban teams need just a few metrics and tools to be effective:

- Cycle time
- Lead time
- Cumulative Flow Diagrams
Cumulative Flow Diagram
To improve software development performance, Kanban teams aim to:
- Shorten cycle and lead times
- Increase throughput
- Improve quality
Where is all the Waste?

- Some see Kanban as being overly concerned with waste elimination.
- Kanban leaders suggest following these Lean priorities:
  - Value trumps Flow
  - Flow trumps Waste Reduction
Recipe for Success

- Focus on Quality
- Reduce Work-in-Progress
- Deliver Often
- Balance Demand against Throughput
- Prioritize

*(Recipe from David Anderson, credited with the first Kanban for software implementation.)*
Summary

❖ Want to try Kanban?
  ❖ Map your **value stream**
  ❖ Establish a **pull** system
  ❖ **Limit WIP** to increase throughput
  ❖ Make it **visible** for your team and stakeholders

❖ **Doing Scrum Now?**
  ❖ Try implementing the material from Corey Ladas’ *Scrum-ban* book. [4]
Thank you!

Interested in Lean or Kanban? Join us for more discussion:
http://groups.google.com/group/limitedwipsocietykc

Troy.Tuttle@adventuretechgroup.com
TroyLTuttle@gmail.com

blog.troytuttle.com
twitter.com/troytuttle
References


(3) David Peter Joyce - [http://leanandkanban.wordpress.com](http://leanandkanban.wordpress.com) and “Pulling Value Lean and Kanban”


(5) Clarke Ching - [www.clarkeching.com](http://www.clarkeching.com)

(6) David Anderson – [www.agilemanagement.net](http://www.agilemanagement.net)