Kanban for Software Engineering

David Joyce

David.Joyce@bbc.com
What I’m Presenting

★ Kanban
★ Lean Software Engineering
★ Features and ROI
★ Ideation Pipeline
★ Metrics
★ Beyond Scrum
★ Q&A
Kanban

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.
Kanban

Anyone that has used a Scrum has used a simple Kanban system.

In Japanese the word Kan means "signal" and "ban" means "card" or “board”.

A Kanban card is a signal that is supposed to trigger action.

Therefore Kanban refers to "signal cards".

You can see Kanban everywhere. The next time you order a drink at Starbucks you can see a Kanban system in place.

The coffee cup with the markings on the side is the Kanban!
Mary Poppendieck had referred to a card wall as a Kanban board in her Lean Software Development book.

While this was technically incorrect* the term Kanban Board had sneaked into the vocabulary of Agile and Software Development and is in usage.

Kanban in manufacturing is the *inspiration* behind what we now call Kanban for Software Engineering.

* as the board didn’t refer to a pull system nor was there a WIP limit
One of the most important facts about Lean is that it is defined as a set of principles, and not as a process that can be replicated across environments.

Today a growing number of practitioners are applying those principles to software engineering.
Managing Engineering

A large proportion of software development problems have very natural workflows that deserve to be managed well.

Its processes can be described in terms of queues and control loops, and managed accordingly.

Time and popularity have exposed some of the limitations of other agile methods in the real world.
Kanban Principles

The basic principles of Kanban for software engineering

- Limit Work in Process (WIP)
- Pull value through (with WIP limit)
- Make it visible (Visual Control)
- Increase throughput
- Fixed Kanban Backlog
- Quality is embedded in (not inspected in)

The team continuously monitor the above to improve
Starting with Kanban

★ Start with what you do now
★ Modify it slightly to implement pull
★ Use a transparent method for viewing work, and organising the team
★ Limit WIP and pull work when the team has capacity
★ Evolve from there by recognising bottlenecks, waste and variability that affect performance

Kanban will support the team’s journey of process improvement and will help expose the best solution.
Value Stream

Identifies all the steps in order to make the desired product.

The word stream implies that there is a smooth, unbroken flow between a sequence of steps.

A continuous smooth flow of valuable, new features, into deployment is the ideal final result.

The value stream includes everybody from the customer to operations and support engineers, and not just development.
Value Stream

Through metrics you can evaluate your efficiency.

How much time spent on value add vs non value add

Quarterly Value Stream Mapping to re-assess the whole value stream.
Kanban Workflow

Lean thinking is generally far more concerned that the right work is being done at the right time, rather than who is doing the work.

People have to work together and they

- might not work at the same speed
- have different skills
- have to synchronise craft work

We organise by task or process, and let team members apply themselves to the workflow in the most efficient manner.
One Piece Flow

The Lean goal of pulling individual work requests through a sequence of value adding activities, quickly and without interruption.

as opposed to

Moving batches of work between stages in a workflow.
Why Pull? Why Kanban?

We:

- 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

What does a Kanban System look like for software engineering?

There is a queue of work, which goes through a number of stages until its done.

When work is completed in a stage, it goes downstream for the next stage.

When someone needs new work to do, they pull from upstream.
**Kanban Pull**

<table>
<thead>
<tr>
<th>Backlog</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step n</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In Process</td>
<td>In Process</td>
<td>In Process</td>
<td></td>
</tr>
</tbody>
</table>

Friday, 24 April 2009
<table>
<thead>
<tr>
<th>Backlog</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step n</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Process</td>
<td>In Process</td>
<td>In Process</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Kanban Pull With Limits

That looks very like a typical Agile Task Board, with maybe a few more steps *

However, there is one more important element which really defines a Kanban system - *limits*.

Because we want to deliver new value quickly, we want to limit the amount of work that we take on at one time.

There are two basic limits - Queue limits and WIP limits.

*Although there is nothing to say there can’t be a single development stage.*
WIP Limits

A limit that governs the maximum number of work items that can be in that state at any instant.

If a state is below its limit then it may take possession of a work item from upstream.

If a state is at its limit, it must wait for one of its own items to be complete, and pulled downstream.
Queues and Queue Limits

A queue (or buffer) between states absorbs variation.

The queue itself has a limit, so that if the queue fills up, the upstream producer will halt.

The queue allows for slack. Optimum flow means just enough slack.

A queue distinguishes work that is eligible to be pulled, from work that is still in process.
Queue and Limits

Backlog

Step 1

Step 2  ...  Step n  Done

Queue  In Process  ...  Queue  In Process

In Process

Friday, 24 April 2009
# Queue and Limits

<table>
<thead>
<tr>
<th>Backlog</th>
<th>Step 1</th>
<th>Step 2</th>
<th>…</th>
<th>Step n</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Queue</strong> (3)</td>
<td><strong>In Process</strong> (2)</td>
<td><strong>Queue</strong></td>
<td><strong>In Process</strong></td>
<td><strong>Queue</strong></td>
<td><strong>In Process</strong></td>
</tr>
</tbody>
</table>

Friday, 24 April 2009
## Queue and Limits

### Backlog

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>...</th>
<th>Step n</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Queue</strong></td>
<td><strong>In Process</strong></td>
<td>...</td>
<td><strong>Queue</strong></td>
<td><strong>In Process</strong></td>
</tr>
<tr>
<td>(3)</td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Friday, 24 April 2009
Queue and Limits

Backlog

Queue
(3)

In Process
(2)

Step 1

Queue
In Process

Step 2

Step n

Done

Friday, 24 April 2009
Queue and Limits

Backlog

<table>
<thead>
<tr>
<th>Queue</th>
<th>In Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3)</td>
<td></td>
</tr>
</tbody>
</table>

Step 1

Step 2

Step n

Done

Friday, 24 April 2009
Queue and Limits

Backlog

Step 1

Queue (3)

In Process (2)

Step 2

In Process

...

Step n

In Process

Done

Friday, 24 April 2009
Lead Time and Cycle Time

Lead time clock starts when the request is made, and ends once delivered.

Cycle time clock starts when work begins on the request, and ends when the item is delivered.

Cycle time is a more mechanical measure of process capability.

Lead time is what the customer sees.

* some teams end once an item is “ready” for delivery.
Limits

We combine the flexibility of craft production with the control of a pipeline.

★ Work in process is limited
★ Cycle Time is managed
★ Its a highly transparent and repeatable processes
★ All the right conditions for continuous improvement
Leading Indicators

Lean scheduling provides crystal clear leading indicators of health.

An increase in WIP today, means an increase in the time to deliver that work in future.

You can’t do more work than you have capacity for, without taking longer to do it.

WIP is easy to control, someone in your business can approve or deny starting new work.

Total WIP is the sum of all parts of your workflow *

* Limiting total WIP still allows for variation in the distribution of WIP between stages.
Leading Indicators

Agile development has long rallied around “inspect and adapt”.

Early agile methods built their feedback around velocity. However this is a trailing indicator.

The regulating power of limits is, it tells you about problems in your process, while you are experiencing the problem!
Before Your Eyes

Queues start backing up immediately following any blockage.

A backed up queue is not a matter of opinion. The consequences are highly predictable.

Shortly after any part of its system goes wrong, the entire system responds by slowing down.

You respond by freeing up resources to fix the problem.
Queuing Theory

*Total Cycle Time = \( \frac{\text{Number of things in process}}{\text{Average Completion Rate}} \)

Example: 1 week = \( \frac{4 \text{ Items}}{4 \text{ per week}} \)

* Little's Law for Queuing Theory
Queuing Theory

To improve Cycle Time

⭐ Reduce Number of Things in Process
⭐ Improve Average Completion Rate
⭐ Reduce rework
⭐ High visibility of blockers and active removal
⭐ Analysis to identify items that are too large

* Little's Law for Queuing Theory
Throughput

Throughput is the rate of delivery of customer valued work into production.

Two major variables regulate Throughput; WIP and Cycle Time.

Throughput allows forecasting of future capability.
Make it Visible!

A Kanban approach, in management terms, is a constant exercise of matching demand with supply, to deliver the right thing at the right time.

You need to see at a glance if we match WIP or not.

The board is a visual control.
Make it Visible!

One could argue this is already done using Agile boards, but at a glance can you see:

- What you are working on?
- If you are overloaded?
- Where bottlenecks are?
- Where gaps are forming?
- What is blocked?

Typical Scrum board
Make it Visible!

Using colour cards and post-its for different type of work, and annotating on cards can quickly enable a team:

- to see what they are working on; those items giving value and those items stopping us from doing so
- improve standups; visual indication of blocked items
- evaluate how good our analysis and QA process is
- do simple tracking; items passed due date, blockers
Make it Visible!

Giving business value:

- A new feature that delivers customer value
- A change request that delivers customer value
- Has a deadline date.

Story
Make it Visible!

Detracting from giving customer value (overhead):

Defects found in live committed to fix.

Defect found against an item in process – attached to the item

Blocker - attached to the item blocked

Requirements clarifications - may come out of a demo or UAT.
A key is stuck up next to the Kanban board.

Recording dates on the cards we can easily obtain metrics to help us improve.
Kanban Board

Does the Kanban board have to look like this?

No, its just an example!

You need to review regularly which states you have.

Start simple and add states as necessary.
What Do I Work On Next?

It's easier to start new work than it is to finish something
"I'm waiting on x"

1. Can you help progress an existing Kanban? Work on that.
2. Don’t have the right skills? Find a bottleneck and work to release it.
3. Don’t have the right skills? Pull in work from the queue.
4. Find other interesting work.

Friday, 24 April 2009
Late binding - you only sign up for something when you have capacity and it enters the queue.
Urgent Work

If there’s an emergency, a support request, a live bug, there’s an empty slot on the board marked “Urgent.”

It doesn’t go through the regular backlog queue.

Strive to finish urgent items quickly, try to keep the slot empty and available at all times.

If the “Urgent” slot is full and another urgent item appears, it has to be added to the backlog queue.
Bugs and Rework

What do we do when we find a bug?
Bugs and Rework

There are 3 solutions:

- Raise a bug ticket, place it in the dev ready queue, mark the item it relates to as blocked
- Stop the line and fix the issue
- Use the emergency queue
Two Tier Kanban
# Two Tier Kanban

<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>
</tbody>
</table>
Two Tier Kanban
Kanban for Everyone!

Kanban doesn’t have to be solely for established or experienced Agile teams.

Kanban offers a path to agility for:

- more traditional teams
- teams in regulated environments
- teams in conservative cultures
- teams with domains that involve lots of specialisation

Kanban has expanded the number of domains that can achieve agility.
Handoffs and Specialisation

Kanban allows for specialisation but does not enforce it.

The team owns the workflow, and knows how best to get it done efficiently.

There has to be a place to put completed work for hand off, when you are done with it.

A completion queue serves this function.
Handoffs and Specialisation

We need to co-ordinate handoffs between ourselves.

The team agrees on what needs to be done prior to a hand off.

In Lean terms this is called standard work.
**Engineering Cycle Time**

We also track Engineering Cycle Time as we are subject to release freezes.

Enables us to independently review and improve this portion of the value stream.
WIP With Pairs

When pair-programming, limit WIP to half the number developers.

A pair could be analyst & developer, developer & tester etc

Each person/pair has 2 slots (can only work on 2 things at once).

The second slot is for critical items that have to be worked on, whilst the other item is marked as blocked.
A New Kind of Standup

The major difference in Kanban standup is that the meeting facilitator enumerates work, not people.

The board shows the status, instead the focus should be on the exceptions.

When enumerating the work it is useful to traverse the board from right to left (downstream to upstream) in order to emphasise pull.
A New Kind of Standup

A Kanban pull system is more focused on work and workflow, than the first generation Agile processes.

Nowhere does this difference in emphasis show more clearly, than in the daily standup meeting.

Kanban projects have no trouble in scaling up to 40 people or more.

Without Kanban, board is ignored or it is woefully out of date, during the standup team members scrabble around updating it.
Kanban Rhythm

Standups

★ Do we have a bottleneck? (look for congestion or gaps in the queues)
★ Do we have a blocker not dealt with?
★ Are we keeping to our work in process limits?
★ Are priorities clear?
★ what we did yesterday, planning today

Post Standup

★ Update charts
★ Remove done items off the board
Bottlenecks

There are two ways to see bottlenecks:

★ Observe things aren’t moving or stalled
★ Observe vacant space on the kanban board
Vacant Space

An example below of a gap in the QA Ready state.

The QA has 3 work items in progress out of a possible 4, however with a gap he may run out of items to pull.
Pipeline Stall

The QA queue is backed up, and the development pipeline is beginning to stall.

Development signals nowhere to move completed work to.

Someone would have to assume the role of QA, and test items that they hadn’t worked on.
Where Is Kanban Working Well?

★ IT Application Maintenance

Examples include Microsoft, Corbis, BBC Worldwide

★ Media Applications

Publishing houses, video, TV, radio, magazines, websites, books. Examples include Authorhouse, BBC, BBC Worldwide, IPC Media, NBC Universal and Corbis

★ Games Production

Games appear to have a lot of specialisation and a lot of hand-offs and kanban helps them manage work in progress and flush out issues quickly.
Putting It All Together

To start match you work in process to your current capacity.

This will be defined by whatever you have least of e.g. analysts or testers.

Take an evolutionary approach and keep changes small and incremental.
Putting It All Together

Try to get people to contribute anything they can towards reducing latency.

Based on the live behaviour of the workflow make hands on adjustments.

If you see a traffic jam forming, close the valve, if you can see an air bubble forming, open it up.

Later you can let a well sized buffer absorb the random variation, without intervention.
Putting It All Together

Maximise throughput of business valued work orders into deployment.

Only time, and real live performance data, can help you to configure correctly.

It make take a while to move enough work through the system, in order to obtain sufficient data.
If you randomly make your target, or consistently miss the target, then the problem is the target is wrong, not the process.

Identify a new target and stabilise the system around it.

Process stability is the necessary condition for continuous improvement.

This kind of thinking is where you cross the line from software development to software engineering.
Two Axioms of Lean Software Engineering

- It is possible to divide the work into small value adding increments, that can be independently scheduled
- It is possible to develop any value adding increment in a continuous flow, from requirement to deployment
Two Axioms of Lean Software Engineering

Ask:

- What would continuous flow of new features look like?
- What would need to happen and in what order?
- What resources would need to be available to make that possible?
- How would roles and responsibilities need to be defined?
- What is the ideal flow?
- What is the best flow we can achieve?
- How can we improve to meet the ideal?
Eliminate Waste!

The 7 Wastes in Lean apply more to manufacturing.

It is much more useful to realise that Waste comes in three abstract types in software engineering

- Rework (or Defects)
- Transaction Costs
- Co-ordination Costs
Cost of Delay

The "waste" metaphor is not proving sufficiently useful in software engineering.

Too many potential "waste" activities also allow some form of information discovery and, hence aren't always waste.

It is more about cost of delay

- Maximising value
- Minimising risk
Cost of Delay Examples

Time from:

- when a bug is introduced, until it is discovered
- when a bug is discovered, until it is fixed
- code being written, until it is integrated
- a misunderstanding in a requirement, until it is discovered
- a requirement released, until it is used
“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

MMF’s can range from very small to very large, depending on context and the stage of the application’s life in the market.

Stories focus on the customer, with business value secondary.

The MMF focuses on the businesses value first, which drives identification of the stories.
Many organisations are now unwilling to tolerate ROIs of more than a year.

This is astonishing considering that 3 - 5 year ROI was the norm just a few years ago.

If short ROIs are only acceptable, how is it possible to release the capital necessary to do software projects?

The answer lies in the re-evaluation of the traditional ROI model for software.
Why use MMFs?

Software products can be deconstructed into units of marketable value.

Typically value is not perceivable as a monolithic whole, but as a series of features.

What is unique about software products is, that features are often separately deliverable.

A complex software product can deliver value even if it isn’t complete.
Why use MMFs?

Typically an MMF creates market value in the following ways:

★ Competitive differentiation
★ Revenue generation
★ Cost saving
★ Brand projection
★ Enhanced loyalty

MMF selection, and ordering, is one of the most important steps in software engineering.
Iterative Funding

Seeing software engineering as the assembly of units of value creation, allows for funding to be made more granular, and more closely aligned to iterative delivery.

This gives rise to the concept of iterative funding.
Ideation Pipeline

Decomposing an idea means

- Analysis activities to understand and flesh-out the idea
- Breaking the idea down into one or more MMFs
- Breaking MMFs into stories and tasks
- Defining acceptance criteria
Ideation Pipeline

We want to manage that flow of work.
Ideation Pipeline

Ideas are placed on the proposed queue.

We move those off by priority into ready for decomposition.

This signals a breakdown session to create MMFs, that the Engineering team will pull.

When an MMF is ready to be pulled from the ideation pipeline, the engineering team breaks it into stories and tasks.
Ideation Pipeline

Once all of the stories are done, the MMF is released, it moves into the Released state on the Ideation board.

We also have a waste state on the board, for items that made it into development, but were never completed as they got cancelled.
Why Metrics?

Metrics are not a tool for managers, they are a tool for everybody.

The team is responsible for its metrics.

Metrics allow for continuous improvement.

Agile teams are now being treated like a business, and held accountable for their results.
What’s the Point?

A team reports we burnt 22 points this sprint!?!?

Sprint goal can be, we did 30 points last sprint, lets do the same...

PMs spend time trying to convert points into days to predict future deliveries

Customers don’t get points.

What is an ideal day?
Real Data

When did you last take into account the entire value stream when estimating?

- e.g. time to deploy, freezes, UAT unavailability

With real data you can make real commitments.
Cycle Time Metrics

Cycle Time data for our T-shirt sized MMFs.

We can use this data when estimating upcoming work.

An additional 10 days to release Large items once built, something to improve!
Throughput Metrics

Total Throughput and Throughput by T-Shirt size.

Here we can see Throughput for the last month.

<table>
<thead>
<tr>
<th>Start Date</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>30-Mar-09</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>23-Mar-09</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td>16-Mar-09</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>09-Mar-09</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>
Cumulative Flow

A simple cumulative flow diagram enabled us to track by state.

Which states you track is up to you.

We can easily view bottlenecks using a CFD.
More Detail Helps

This is the typical “Not Done” - “Doing” - “Done” style.

We aim to narrow the “Doing” gap using WIP limits.
Cumulative Flow

This example presents more detail by state across the Value Stream.

You can see we have items stacking up that aren’t released, due to release freezes.
Cumulative Flow

Cause and effect; the amount of work in process, and its effect on the time to deliver that work.
Work Breakdown

It is useful for a team, and their customers, to know what kind of work is being completed, and how much is providing customer value.

Changes to our Kanban cards achieves this. We record in a spreadsheet.
Work Breakdown

The number of features is on the rise, blockers are under control, and less technical features are being implemented 😊

However live bugs that we had to fix is on the rise 😞
The Parking Lot

A high level view of progress of each MMF.

Completed component stories against total is shown.

<table>
<thead>
<tr>
<th>Name</th>
<th>Started</th>
<th>Completed</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>100.0%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>100.0%</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>100.0%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>100.0%</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>8</td>
<td>88.9%</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>100.0%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>100.0%</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>100.0%</td>
</tr>
<tr>
<td></td>
<td>56</td>
<td>54</td>
<td>96.4%</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>100.0%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>24</td>
<td>100.0%</td>
</tr>
</tbody>
</table>
New Approach

What I have shown so far can be achieved using Kanban and Scrum.

Can we go further?
Scrum

One of the origins of Scrum is “The New New Product Development Game” published in 1986. Three approaches were described.

★ Type A - typical Scrum, single phases proceeding sequentially.
★ Type B - overlapping at the edges.
★ Type C - multiple overlapping work, where each is variable length and not time boxed.

Type C is described as suitable for more mature teams, is akin to Kanban.
Scrumban

Scrumban is useful for existing Scrum teams who are looking to improve their scale or capability.

Developed in 2004 at Microsoft by Corey Ladas.

It's possible to incrementally enhance Scrum

- Level 1 - Implementing a simple Kanban pull system, limiting work to capacity, managing specialisation and flow
- Level 2 - As cycle time become the focus of performance, limiting the iteration backlogs and estimations
- Level 3 - de-couple the release and planning periods
- Finally pull prioritised work on demand
Kanban Retrospectives

We have more choice on when and how to reflect and improve.

- Regularly scheduled Retrospective as with Scrum
- After a feature has been deployed
- When we have to “Stop the line”
- Weekly mini retrospective
  - Look back at the last week [max 5 min] (What happened? Are we satisfied? Should we adjust WiP limit?)
  - Team picks one thing to improve on for upcoming week [max 2 min]
  - Write this improvement goal down on top of Kanban board
Kanban Retrospectives

Don’t rush in and eliminate retrospectives. Don’t proscribe types of retrospectives.

Let the team make its own decision when it is ready, and embrace the evolution that comes with continuous improvement.
Striking a Different Bargain With The Business

Scrum’s expectation of the business is:

You may not interrupt work in process (during a sprint), and you may not adjust the work plan more frequently than once every $x$ days (sprint planning).

A scope-driven goal is only one kind of goal.

Another kind of goal is quality of service, and that is what we do.

The engineering team’s promise is:

When we agree to take on a work request, we intend to deliver it within $x$ days.
A New Kind of Planning

Rather than planning a timebox worth of product increment, the team can plan a single MMF at a time.

The team limits WIP in order to minimise the cycle time of MMFs.

The planning is ‘de-coupled’ from the release.

A commitment and deadline can be made per MMF.
A New Kind of Planning

No customer ever comes asking for “4 weeks worth of code”.

They want features and problems solved.

Timeboxes break up work into chunks that have nothing to do with value.

MMFs directly solve the problem that timeboxes merely work around.
Different planning buckets for different time horizons:

- **6 week bucket**: well-defined MMFs
- **3 month bucket**: loosely-defined features
- **6 month bucket**: broad feature areas
- **1 year bucket**: strategies, goals, market forces

Update ALL of the buckets once every 6 weeks OR every time an MMF completes.
Rolling Wave Planning

- Small, well-defined work items
- Larger functional goals
- Specific problems to solve
- Strategic product goals
- Strategic product line goals

A Scrum Sprint
Dimensional Planning

The 3 well known dimensions are Time, Resources and Scope, we introduce another dimension: **Depth**.

If applied well it will increase project throughput, and shorten the feedback loop.

For the different depths we use the following levels:

- **dirt road**: This level is the minimum implementation with manual workarounds. It’s recognised that it will have a limited life span.

- **cobbledstone road**: This level is the bare minimal implementation, but the foundations have been laid for a longer term solution.

- **asphalt road**: This level is the full implementation.
The descriptions seem to resonate with the customer, and leads to good discussion.

Quote for each option via T-Shirt sizes and Cycle Time metrics.

Build options can be weighed against potential increased revenue or cost savings.
Releasing

The rate of features exiting the process has to be the same as the rate of features entering.

Some features are bigger, some are smaller, but on average most releases will be of similar size.

The release interval should be set to the optimum point between cost of deploying vs the opportunity cost.

The ideal is deployment on demand.
Quarterly Commitments

Time boxes and sprint goals are a substitute for having small, well-defined business goals.

- Forecast quarterly goals and objectives
- Prioritise MMFs to meet those goals and objectives
- Release regularly
- Provide metrics
- Build trust that the team is working to its full capacity
- Continuously improve
Quarterly Goals

The Kanban board can be updated with two new states; Goals and Objectives.

Limits on each have been created for the quarter.

The breakdown is Business goal-> Objective-> MMF -> Story

The team asks themselves:

★ When we finish this story, would the feature be complete?

★ When we finish this feature, is the epic complete?

★ When we finish this epic is the objective complete?
Planning on Demand

Agility implies an ability to respond to demand.

The Kanban backlog should reflect the current understanding of business circumstances.

The ideal work planning process should provide the team with the best thing to work on next, no more, no less.

We should look to limit our Kanban backlog*

*You can still have a total backlog, see ideation pipeline
Order Point

An order point facilitates scheduling of planning activities.

As we pull items from the backlog it will begin to diminish, until the number of remaining items drops below the order point.

When this happens a notice goes out to organise the next planning meeting.
The Case Against Iterations

Iterations will become less useful. Whether iterations are useful is up to your team.

The starting point is to retrospect. Each team and circumstance will be different.

Nothing in the Agile manifesto says you have to use iterations, standups, planning games etc etc

However nothing in Kanban says you have to drop all Scrum ceremonies.

Use whatever works for you!
Kanban vs Scrum Similarities

★ Both are Lean and Agile
★ Both use pull scheduling
★ Both limit WIP
★ Both use transparency to drive process improvement
★ Both focus on delivering releasable software early and often
★ Both are based on self-organising teams
★ Both require breaking the work into pieces
★ Release plan continuously optimised based on empirical data (velocity / lead time)
# Kanban vs Scrum Differences

<table>
<thead>
<tr>
<th>Scrum</th>
<th>Kanban</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeboxed iterations prescribed</td>
<td><strong>Iterations optional.</strong> Can have separate cadences for planning, release, and process improvement. Can be event-driven instead of iterative.</td>
</tr>
<tr>
<td>Team commits to a specific amount of work for this iteration.</td>
<td><strong>Commitment optional.</strong></td>
</tr>
<tr>
<td>Uses <strong>Velocity</strong> as default metric for planning and process improvement.</td>
<td>Uses <strong>Lead time</strong> as default metric for planning and process improvement.</td>
</tr>
<tr>
<td><strong>Cross-functional teams</strong> prescribed</td>
<td>Cross-functional teams optional. <strong>Specialist teams allowed.</strong></td>
</tr>
<tr>
<td><strong>Items must be broken down</strong> so they can be completed within 1 sprint</td>
<td>No particular item size is prescribed.</td>
</tr>
<tr>
<td><strong>Burndown chart</strong> prescribed</td>
<td>No particular type of diagram is prescribed</td>
</tr>
<tr>
<td><strong>WIP limited indirectly</strong> (via sprint plan)</td>
<td><strong>WIP limited directly</strong> (per workflow state)</td>
</tr>
<tr>
<td><strong>Estimation</strong> prescribed</td>
<td><strong>Estimation optional</strong></td>
</tr>
<tr>
<td>Cannot add items to ongoing iteration.</td>
<td>Can add new items whenever capacity is available</td>
</tr>
<tr>
<td>A sprint backlog is owned by one specific team</td>
<td>A kanban board may be shared by multiple teams or individuals</td>
</tr>
<tr>
<td><strong>Prescribes 3 roles</strong> (PO/SM/Team)</td>
<td><strong>Doesn’t prescribe any roles</strong></td>
</tr>
<tr>
<td>A Scrum board is <strong>reset</strong> between each sprint</td>
<td>A kanban board is <strong>persistent</strong></td>
</tr>
<tr>
<td>Prescribes a prioritized product backlog</td>
<td>Prioritization is optional.</td>
</tr>
</tbody>
</table>
In Conclusion

Lean production is probably the single greatest enabler of continuous improvement.

Successful implementation of a lean production line is likely to yield a dramatic boost in the first year, as capacity is balanced against demand, and the easily identified waste is removed.

A 100% first year productivity improvement would not be unexpected, from a successful launch.
In Conclusion

Kanban is about

- Using virtual signal cards to limit WIP
- Focusing teams on quality and cycle time
- Balancing demand against throughput
- Prioritising appropriately

We also discover that it facilitates some cultural changes, and the development of high maturity organisational behaviours.
In Conclusion

You can keep doing many aspects of your current process, but I can't see who wouldn't want to:

- Limit WIP and in doing so see improved throughout
- Make work visible to the team, other teams and management
- Have better metrics rather than burn down
- Have leading indicators such as queue utilisation and bottlenecks
- Have better management of handoffs and specialisation
- A more meaningful conversation with customers on ROI, capacity and capability
In Conclusion

You may find once you have achieved this, you start to question aspects of your current process, which is all part of continuous improvement.

The beauty is that Kanban isn’t prescriptive, your situation will be different to everyone else, so do what works for you!

Don’t limit yourself to one tool!

Mix and match the tools as you need! I can’t imagine a successful Scrum team that doesn’t include most elements of XP.
More Information

My blog http://leanandkanban.wordpress.com/

David Anderson http://www.agilemanagement.net/Articles/Weblog/blog.html

I would also highly recommend Corey Ladas Scrumban book and blog http://leansoftwareengineering.com

Karl Scotland is creating an excellent how to http://availagility.wordpress.com/2008/10/28/kanban-flow-and-cadence/

Software by Numbers book by M Denne & H Cleland-Huang

Dedicated Yahoo! Kanban group

http://groups.yahoo.com/group/kanbandev/
More Information

David Anderson visit May 1st.

May 19th - “In the Brain of Evening Session” on Kanban at BBC Worldwide

Lean and Kanban conference in Miami 6th - 8th May

Lean and Kanban conference in London 27th - 29th September

Course being written at Skills Matter look out for pre-teach sessions.
Thank You!

Any Questions?
References

[9] Rob Bowley and Matt Wynne - Evolving from Scrum to Lean
[10] Software by Numbers by Mark Denne and Jane Clelenad-Huang
Additional Info
Japanese Words

I've completely lost interest in Lean in the sense of analogously mapping Lean techniques or ideas, and that I am eliminating all use of Japanese words with the exception of Kanban and Kaizen from my material.

I've decided to title my key note for Lean & Kanban 2009 as "Forget the Japanese Words! Focus on Culture & Risk Management"
Lean Software Engineering Manifesto?

Lean Software Engineering will:

⭐ Develop a deep characterisation of customer needs, based on close customer interaction, and sophisticated modelling tools

⭐ Produce finely grained, formally specified requirements, that can be independently scheduled for development

⭐ Follow a rigorous and formal engineering workflow, with multiple preventive quality control steps, and planned continuous process improvement
Lean Software Engineering Manifesto?

- Collect useful statistics about quality and productivity, as an integrated part of everyday work activities
- Continuously integrate new features into a working, stable, secure, and reliable system
- Make those new features available to customers, at every appropriate opportunity
Kanban Overview
There is no Kanban Process

This is not a prescription. There is no “kanban process” as such. The process is whatever the team or organisation is currently using. The technique is to map that process, put WIP limits on it and start a pull system. Everything else evolves from there.

You need to get away from agile process as prescription and start to recognise that every situation and every project is different and that process must be described as sets of policies and those policies should be chosen, or modified to fit the risk profile and specific circumstances of the organisation and its value chain.

The whole point of kanban is to enable people to find their own optimisations and change their own process, not to impose a new way of working upon them and ask them to change their behaviour. Kanban enables a situationally specific process to evolve rather than the imposition of an unsuitable prescription that was developed for someone else, in some other value stream, in some other organisation, on a project with a different set of resources, budget, schedule and risk profile.

I want to prevent Kanban (in software engineering) becoming a dogmatic superstitious practice level process. What is different about Kanban is it does not specify engineering, workflow or project management practices. These are left to emerge in a situational/context specific fashion. In this respect it is completely different from any other agile approach previously published.

Kanban is about the notion that “your situation is truly different” and “we will not impose a process upon you”.
Managing Policies

Kanban focus us on creating and managing policies:

- Classes of service (prioritisation)
- Managing bottlenecks (throughput)
- Co-ordination (waste)
Multi Vote Kaizen Board

A Kaizen “improvement” board.

Contains technical debt stories, retrospective actions, non critical bugs, nice to haves, time savers, utilities etc

During slack time an item can be pulled from the Kaizen board.

The question is which one should we pull?

Using multi-voting the team can vote on which item is the next highest priority to be pulled once a slot becomes available.
MMF Owner

One person owns an MMF and that person promiscuously pairs with other members of the team (including the Architect).

This allows code knowledge sharing and new input into something that is being developed, it also works well with a rotating support model.

The minimum time the MMF owner spends with their pair will be defined by the team, it could be a few days or a week for example.

Most of our streams of work are developed by a pair.

However on occasion there will be small items of work that can be completed by one developer, for these items its they will be regularly reviewed with the Lead Developer and Architect.
Show and Tell

The aim is to perform a show and tell to the QA and BA once each item of work has been completed (in the developers eyes) and is ready to move further in the pipeline.

This is the first round of testing and will avoid rework where an implementation may have missed the mark, even if the acceptance criteria has been met.
Why Queue?

The irregularity of requirements and the creative, knowledge intensive nature of a design activity like software development rules out clocked workflow synchronisation.

Risk and uncertainty are built into the nature of development work.

Novelty is what gives software its value, so we can only get so far in reducing this kind of variation before you have to mitigate and adapt to it.
Consumers Pull Value from Producers

- You, the producer, give me, the consumer, an empty Kanban
- I, the consumer, fill out the Kanban and give it to you, the producer
- You make the thing on the Kanban and give it back to me
- I verify that the thing you made me is what I wanted
Focusing Solely on WIP is Wrong

Many people have asked the same question, if you have a spare slot upstream and nothing is blocked downstream then should we pull new work into the pipeline. You would think the obvious answer to this is YES, but doing so increases your Work in Process (WiP).

The first ever Scrumban team ran into this issue at Corbis, and they switched to releasing team members to start work on other features.

Personally, I don’t think that there is a right or a wrong answer to this. You let the data make the decision for you. If releasing people to start something new improves the key indicators of throughput, cycle time, and initial quality (defect rate), then why wouldn’t you do it. Focusing solely on minimising WIP is wrong.

In Lean operational decisions, value trumps flow, and flow trumps waste reduction.

By focusing solely on WIP you are focusing solely on waste reduction. If flow and/or value can be improved by (slightly) increasing WIP then you should do it. In order to later reduce WIP without impacting the improved flow or value delivery, you need to reduce variability by improving analysis techniques.
Handoffs and Specialisation

If we are going to allow workflow specialisation and the handoffs that result then we need some agreement about what results to expect at each handoff.

We can do that by defining some simple work standards or standard procedures for each state.

These can be drawn directly on the task board.
“In lean enterprises, traditional organisational structures give way to new team-oriented organisations which are centred on the flow of value, not on functional expertise.”

Mary Poppendieck
What is Lean?

Lean is about improved quality, efficiency, reduced costs.

Flow improvements and waste elimination’s are acts that respond to observable events and statuses in order to achieve the goals.

Avoid the injection of work that can’t be processed by the system or avoid the injection of work that we don’t need right now.

In both cases, you will be just accumulating inventory, slowing down the system and the continuous improvement process.

To do that, you need to en-queue the work and be sure that the next work is the most important one at the moment.
I don’t consider Agile to occupy any privileged position within software development methodology. It is one narrowly conceived and applied subset of iterative/incremental software development, which itself is a subset of software development methodology overall, which in turn is a subset of systems engineering and product development methodology. [2]

The definition of “agile” consists of nothing more than the values and principles expressed in the Agile Manifesto. The authors wisely chose to remain silent about matters such as batch-and-queue operations, iterations, retrospectives, and other details of popular methodologies, thus leaving the door open for creativity, innovation, and improvement. By the same token, there is nothing in the kanban approach that conflicts with any of the values or principles of agile development. [3]

Often objections of Kanban are based around the notion that the presented approach is “not agile.” Meaning, its not Extreme Programming or Scrum. It’s rather a pity that this comes up because it is the objector who is missing the point. The entire point is that Kanban enabled teams who were not using Extreme Programming or Scrum to make significant progress. By adding kanban discipline to their world, these teams “uncover[ed] better ways of developing software.” [1]
“A regular cadence, or ‘heartbeat,’ establishes the capability of a team to reliably deliver working software at a dependable velocity. An organisation that delivers at a regular cadence has established its process capability and can easily measure its capacity.”

Mary Poppendieck
Most agile teams are familiar with the following template when writing a user story:

As a [role]
I want a [feature]
so that [goal]

This format is widely used and has proved very popular. The level of detail written on a card varies as does acceptance criteria etc. A card only has so much space though so at some point you have a conversation with the customer about the requirement which is its purpose as a conversation placeholder.

In the advent of MMFs it seems a little strange to put something as important as the goal last when using this format. We strive to define the value a feature will bring when it is completed, as this assists us with planning and prioritisation.

A new format has been suggested for MMFs:

In order to [goal]
[stakeholder] needs [feature]

The point here is the order. The value (goal) comes first, and not last.
Epics and Feature Sets

**Epic - Build an airplane.** Value is getting from a to b.

- MMF1 - Take off
- MMF2 - Fly
- MMF3 - Land

Each MMF gives customer value and would have a lot of stories within them to complete. The thing to note here is that overall none of them released independently will be of value to the customer, only once all three are complete could we release as its no good being able to take off if you cant land. Therefore the three together are classed as a Minimally Useful Feature Set.

We can release each independently for testing and to mitigate risk but its important to be aware that not every MMF will give instant value to the customer.

Carry Passengers and Carry Bagage MMFs are not necessary but the customer might demand them before they sign-off.

*Determine the minimum deployable feature set and then build that as fast as you can. When you cant take anything else away without making the system clearly nonsensical then you’ve found the minimum feature set.* [2]
T-Shirt Sizing

Do all items that enter the queue have to be the same size?

In an ideal world, all items would be of similar size, which will give you a single, reliable cycle-time and throughput.

However, it's not critical. You can categorise items by T-Shirt sizing them (S, M, L) to get a range of cycle times and use this in your planning.
Measuring Quality

Measuring progress against budget is fairly well defined, however something as subjective as measuring quality is much tougher.

Our teams have been asked to figure out indicators of quality on a project. The aim is to ensure that we have an upward trend for quality and that Lead Developers actively work with their developers to ensure this trend continues in the right direction.

The following measures were agreed upon:

**Cycle Time/Time to live (goal gets quicker)**

This is the time it takes for an item to be put on the board for development to when it goes live. If this is taking a long time and/or is taking longer over time then this points to a problem with quality i.e. that making changes or additions are becoming harder to implement.

To track note the start and end date on every card and compile a total each week.
Measuring Quality

Frequency of releases (goal minimum every 2 weeks)

The more often work is released the less risk there is to the project that something new introduced to live will break the system. A target of releasing every 2 weeks at a minimum (barring release freezes) should be adhered to. Ideally as soon as something is ready that can give customers increased value it should be released, rather than waiting 2 weeks. The more often you release points to more adaptable code, a well defined release process and smooth flow. To track record how often a release occurs per week.

Throughput/number of items released (goal upwards trend)

The number of items released per iteration/week gives an indication of the size of stories a team is working on. If they are too large then this is shown as a team not releasing many items. Large stories can become too complex and incur a higher level of risk. To track record the number of items released per iteration/week.
Measuring Quality

**Check in size/frequency per repository (goal small in size, high in frequency)**

Work should be checked often and be small in size. Work that is held outside of the repository for a long time or that is large in nature increases risk. To track use Subversion reporting.

**Unit test coverage trend weighted across all solutions (goal upwards trend)**

We are already tracking test coverage at the solution level, an aggregated and weighted (taking into account code that has been touched recently vs legacy code) summary needs to be created. Track using NDpend.
A Happy QA

When we were doing traditional Scrum our board was often overloaded in the QA lane, cards would stack up as developers were churning out work at a high rate and our 1 tester was seriously overworked.

Scrum promotes the notion of an agile team self organising and pulling together, however this doesn’t happen as often as one would like.

After setting limits on our Kanban board the team soon began to realise that they could no longer stack work up, once a limit had been reached there was no where to go and the line would become blocked, so they started assisting the QA and testing work that they hadn’t written.

Several months in we now have more manageable board, the QA isn’t overworked and the team pull together like never before.
Scrum in a Nutshell

- Split your organisation into small, cross-functional, self-organising teams.
- Split your work into a list of small, concrete deliverables.
- Assign someone to be responsible for that list and to sort the list by priority.
- The implementation team estimates the relative size of each item.
- Split time into short fixed-length iterations (usually 1 – 4 weeks), with potentially shippable code demonstrated after each iteration.
- Optimise the release plan and update priorities in collaboration with the customer, based on insights gained by inspecting the release after each iteration.
- Optimise the process by having a retrospective after each iteration.
The Case Against Iterations

A Kanban approach includes all the elements of Scrum (including stand ups as well), but decouples them from all being tied to the Sprint. This can create a more natural process.

Steve Freeman

Kanban development systems as currently practised certainly use iterations, but they use them in the original meaning of the word rather than the hijacked Agile definition.

Timeboxes are at odds with continuous improvement. If you want continuous improvement, you have to start with a continuous process. The significance of flow is about creating the conditions where continuous improvement is actually possible.

Burning your backlog down to zero is bad for productivity and usually bad for quality (ask Deming why)

Timeboxes impose an arbitrary process artefact on the customer. No customer ever comes asking for “4 weeks worth of code”. They want features and problems solved.

Timeboxes break up work into chunks that have nothing to do with value. MMFs directly solve the problem that timeboxes merely work around.

Corey Ladas
Scrum is more prescriptive than Kanban.

Agile methods are sometimes called lightweight methods, specifically because they are less prescriptive than traditional methods.

In fact, the first tenet of the Agile Manifesto is “Individuals and Interactions over Processes and Tools”.

Scrum and Kanban are both highly adaptive, but relatively speaking Scrum is more prescriptive than Kanban.

Scrum gives you more constraints, and thereby leaves less options are open. For example Scrum prescribes the use of iterations, Kanban doesn’t.
Kanban Team Examples

Kanban team #1 (single cadence): “We do Scrum iterations”

Kanban team #2 (three cadences): “We have three difference cadences. Every week we release whatever is ready for release. Every second week we have a planning meeting and update our priorities and release plans. Every fourth week we have a retrospective meeting to tweak and improve our process”

Kanban team #3 (mostly event-driven): “We trigger a planning meeting whenever we start running out of stuff to do. We trigger a release whenever there is a MMF ready for release. We trigger a spontaneous quality circle whenever we bump into the same problem the second time. We also do a more in-depth retrospective every fourth week.”
Scrum
WIP Limits

Kanban limits WIP per workflow state, Scrum limits WIP per iteration
There were a number of things that were bothering me about Scrum:
I wanted to change the backlog more often than the timebox allowed
At any given moment, only one item in the backlog needs to be prioritised. Further prioritisation is waste.
I wanted a specific mechanism to limit multitasking
I hated estimating
I hated negotiating Sprint goals
Sprint planning implicitly encourages people to pre-commit to work assignments
The ScrumMaster role is prone to abuse and/or waste
Burndowns reek of Management by Objective
Preposterous terminology
The thing that I wanted most was the smoothest possible flow of pending work into deployment, and Scrum just didn’t give me that. So, I proposed:

A daily standup
A single (roughly) prioritised backlog
Each person on the team is responsible for exactly two work items by the end of any standup
Every work item is associated with a workflow, and work item status is indicated by workflow state
A work item requires some kind of peer review and approval in order to be marked complete
New items can be added to the backlog at any time
There is a regular project review
The backlog must be regularly (but minimally) re-sorted
Status reporting is by cumulative flow only
Scrum Frustrations

Customer/Product Owner get frustrated that they have to wait for a sprint to end before new work will be considered. With Kanban you can dispense with sprints. As soon as 1 of those items is complete they pull in the next priority item from the queue. Therefore the wait time to inject a new request is minimal.
Planning meeting attendance is poor as the customer doesn’t have the time to spend 2 hours every 2 weeks planning.

You dispense with the ceremony of sprint planning, you plan only when you need to (when the queued upstream work is running short) and only for as long as you need to get enough new items in the queue to keep things flowing.
Scrum Frustrations

Sprint planning would last for a couple of hours, everyone would leave the session happy to what they have committed to. Within a couple of days an urgent support issue or 2 would come into the team and our planning was thrown, we would try and adjust then someone would be pulled onto another project for a hotfix or someone would go sick, our plans soon became meaningless and this would frustrate the team.

We now plan when you need to (Just in time planning) based on the need for more items to fill a queue that is drying up. You dont try to predict the future!
Scrum Frustrations

If an item is complete why should I wait until the end of the sprint to start using it?

If your release process allows you can release as soon as something of value to the customer is ready. Release planning becomes smaller, easier and less risky.