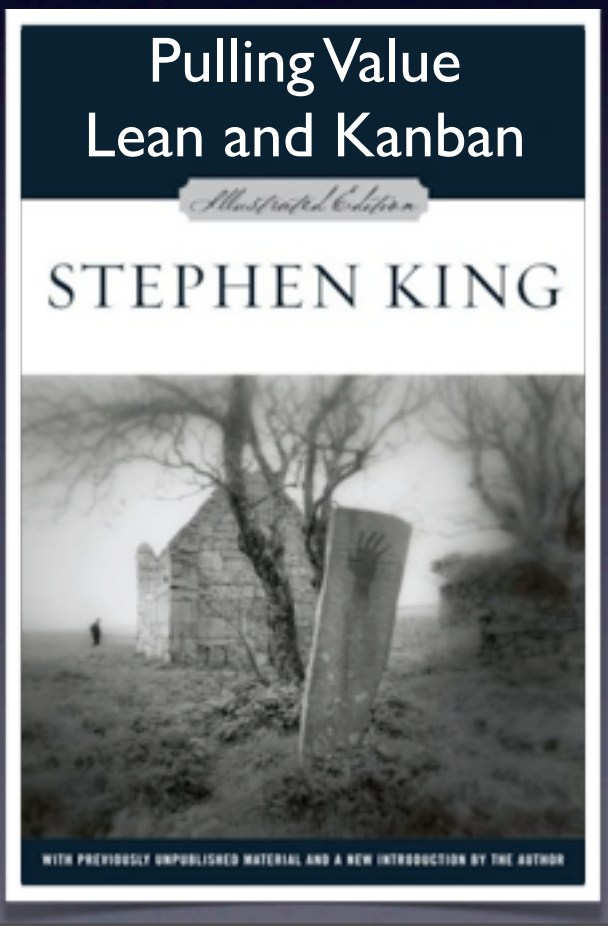
Pulling Value Lean and Kanban

David Joyce



David.Joyce@bbc.com

What I'm Presenting

- Lean and Kanban for Software Engineering
- Managing Risk and Maximising Value
- Features and Return On Investment
- 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.**

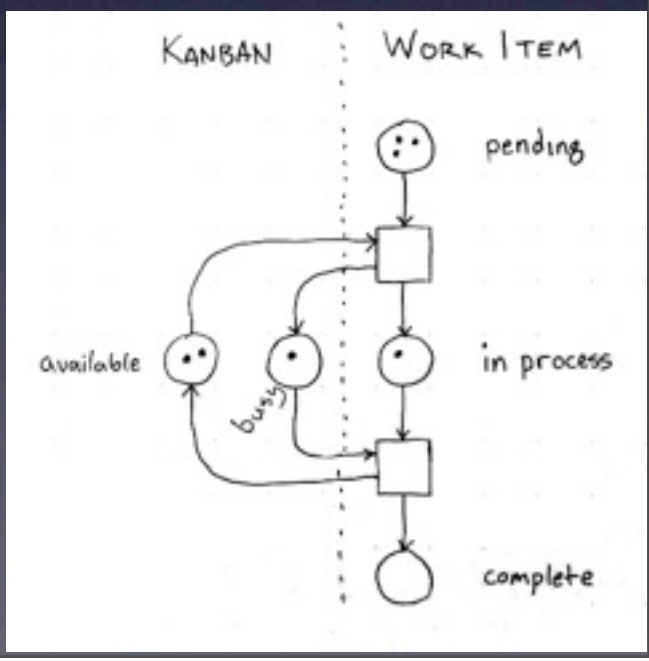
[3]

Managing Engineering

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

Time and popularity is exposing some of the **limitations** of some popular **agile methods** in the **real world**.

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



[2]

Looking to Lean Principles

Lean 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**.

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

Kanban is a true pull system implementation in software engineering.

[2]

Pillars of Lean

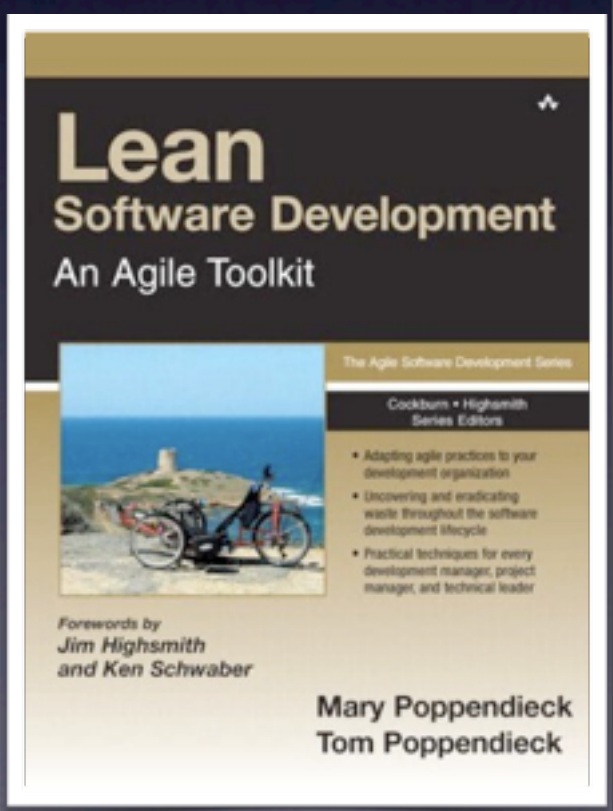
- **Pull**
- **Continuous Flow**
- **Customer Value**
- **Waste Elimination**
- **Continuous Improvement**

Kanban **fully implements** all 5 pillars.

Kanban

Mary Poppendieck had referred to Kanban in her **Lean Software Development book**.

While this was technically incorrect* the term Kanban Board had sneaked into the **vocabulary of Agile**.



[1] * as the board didn't refer to a pull system nor was there a WIP limit

Kanban Principles

- Agree a team **capacity**
- **Limit WIP** (Work in Process) to that **capacity**
- **Pull value through** the Value Stream
- Make both **work** and **workflow visible**

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

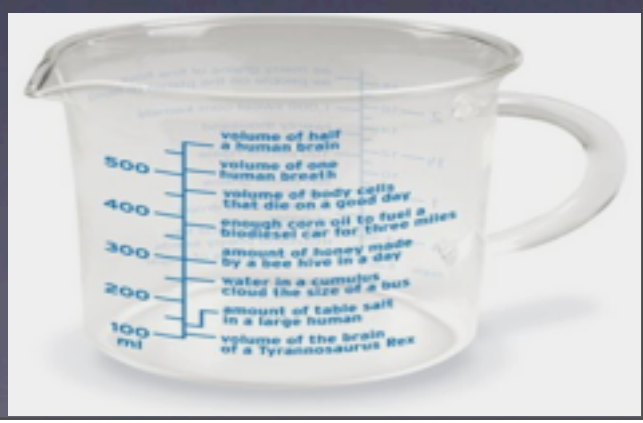
[1]

Capacity and Limits

Through **agreement** with our **value chain partners** we **establish**

- Our **capacity** on what is a fair, and **reasonable** expectation for **workload** on our team.
- A reasonable set of understandable **working policies**.

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



Work In Process

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

We want to **finish** items before **starting** others.

Context **switching** costs **20%** productivity.

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

[2]

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.

[3]

Continuous Flow

We ask:

- What would **continuous flow** of **new features** look like?
- What would need to happen and in **what order**?
- What **resources** are needed 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**?

[2]

Value Stream

Identifies all the **steps** in order from **idea to delivery**.

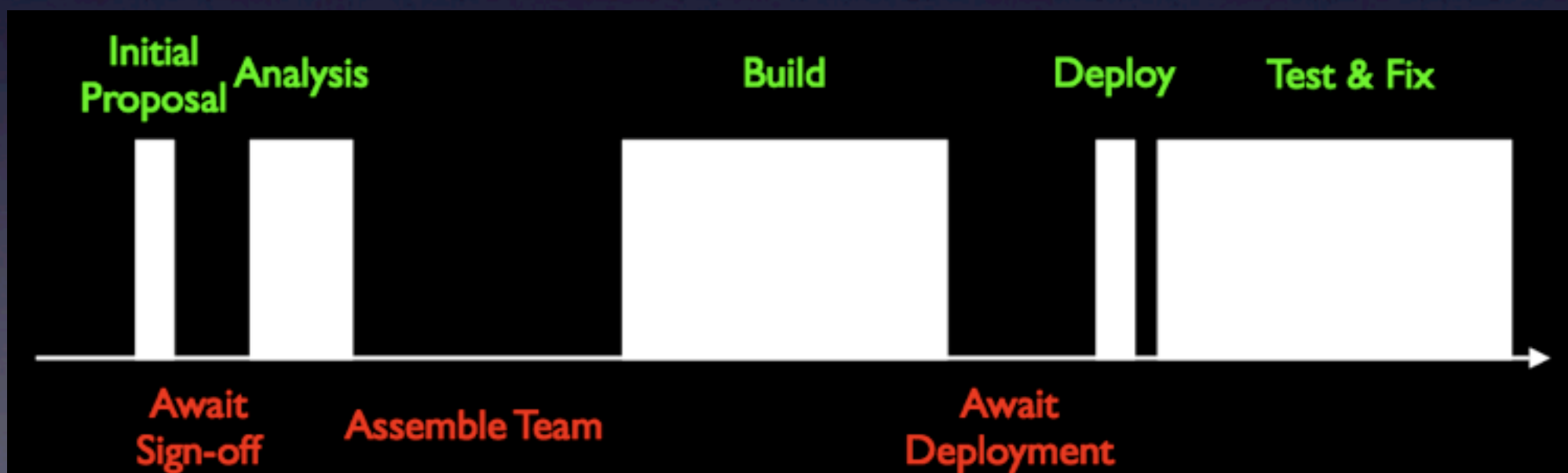
A continuous smooth **flow** of **valuable**, new **features**, into deployment.

The value stream includes ***everybody*** from the **customer** to **support**.

Value Stream Mapping

How much time is spent on **value add** vs **non value add**

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



[9]

Pull Work Not Push

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**.

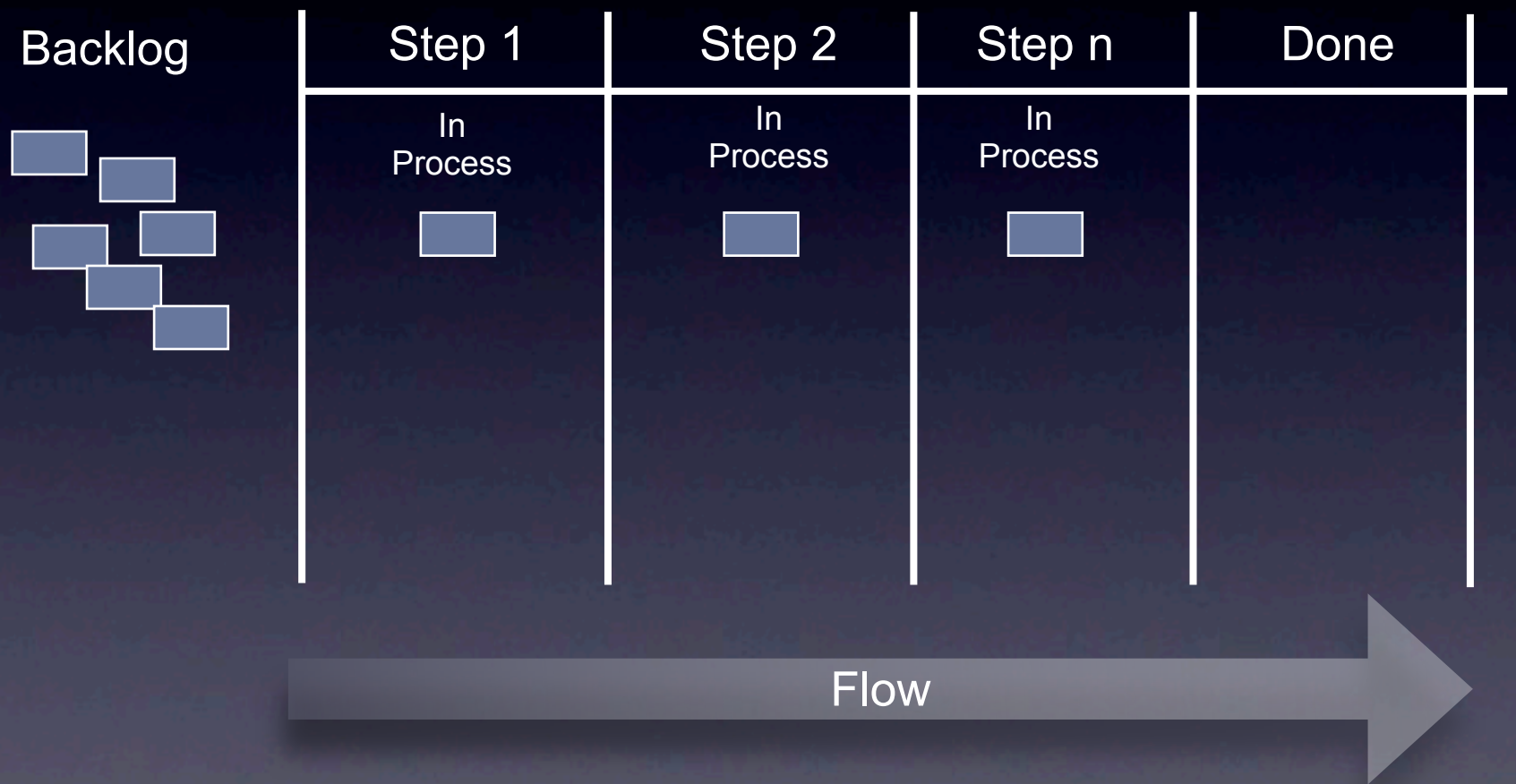
Why Pull? Why Kanban?

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**

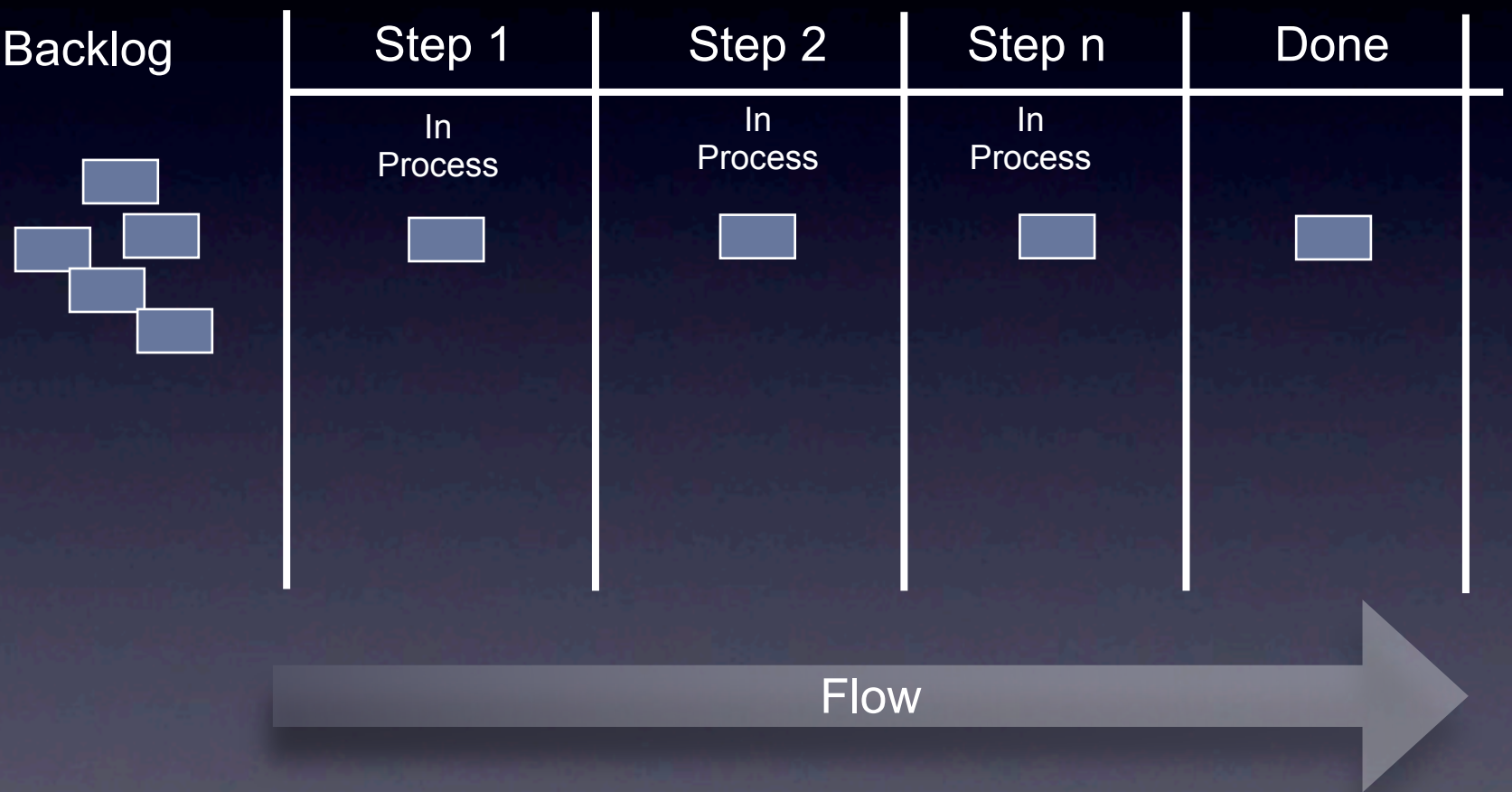
[2]

Kanban Pull



[3]

Kanban Pull



[3]

Kanban Pull With Limits

That looks very like a **typical Agile Task Board**.

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

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

[3]

WIP Limits

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**.

[2]

Queues and Queue Limits

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

A queue (or buffer) between states **absorbs variation**.

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

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

[2]

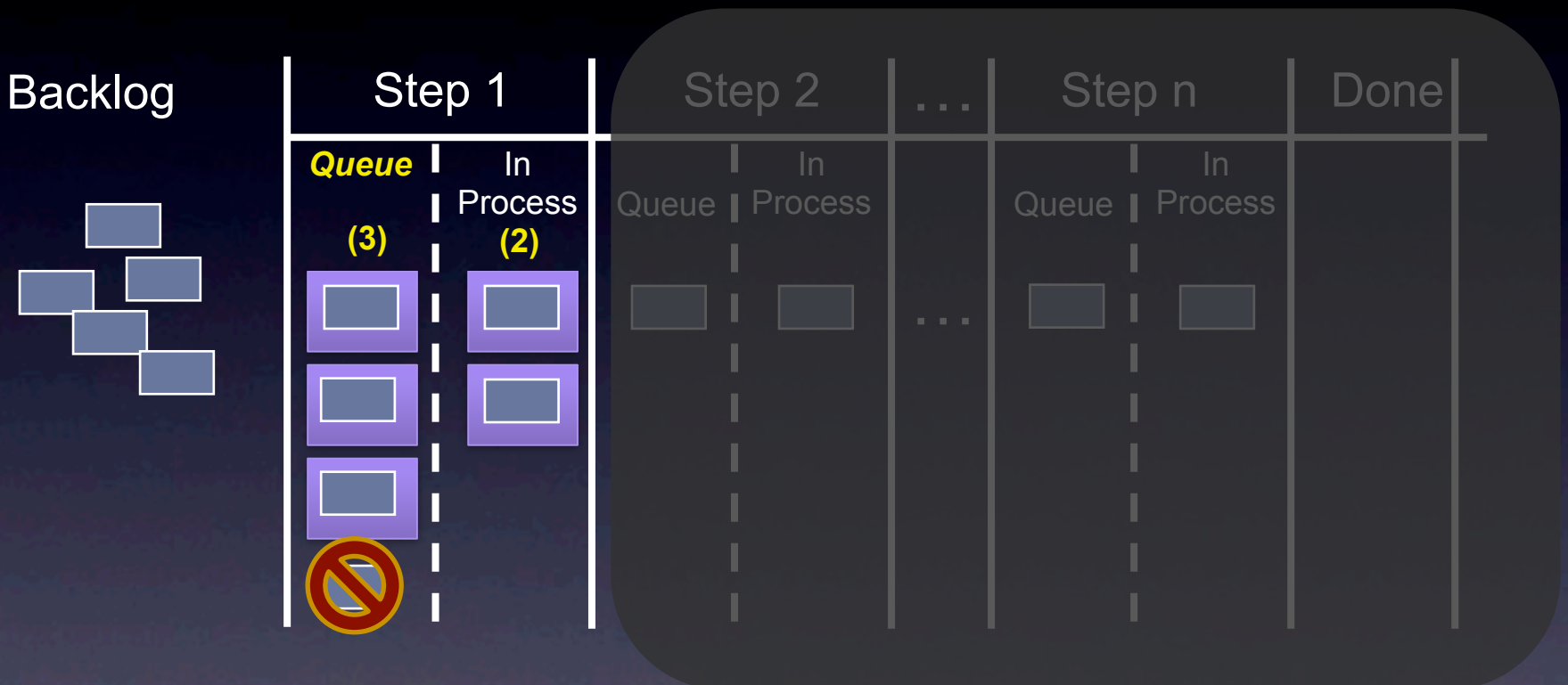
Queues and Limits

[3]

Queues and Limits

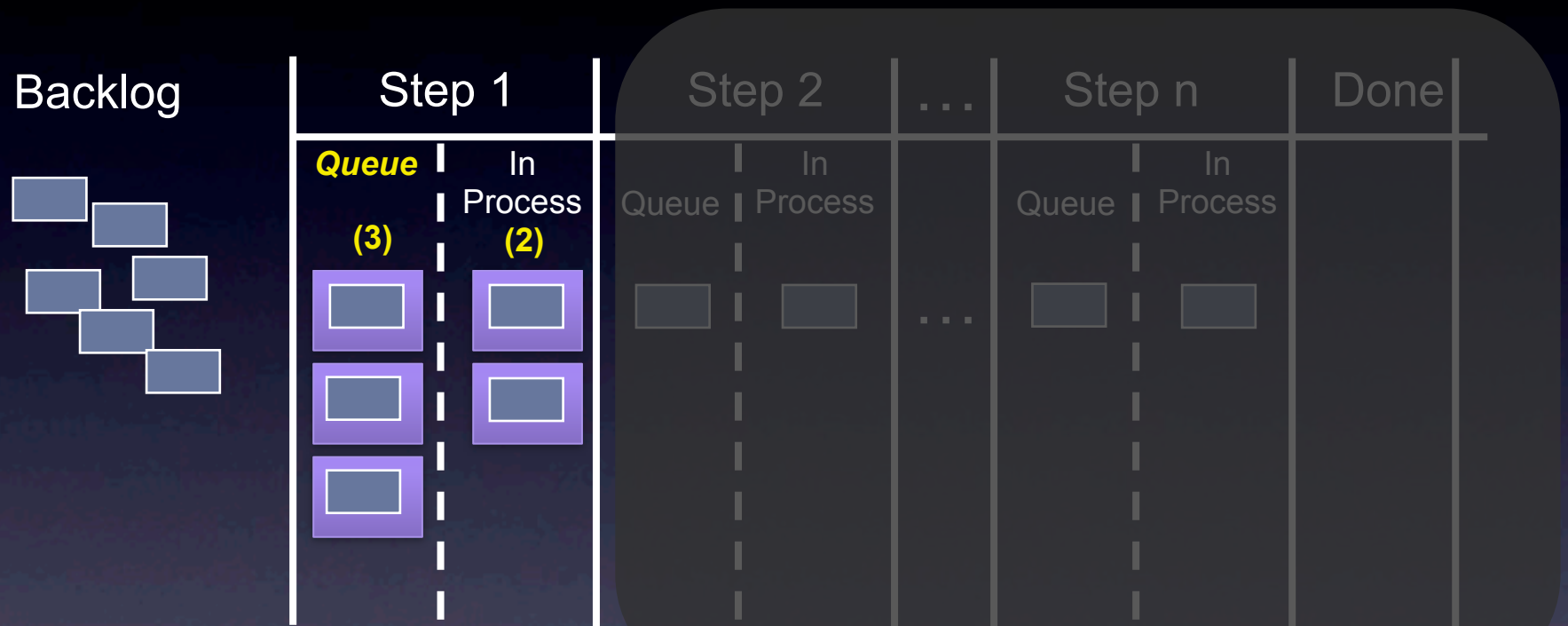
[3]

Queues and Limits



[3]

Queues and Limits



[3]

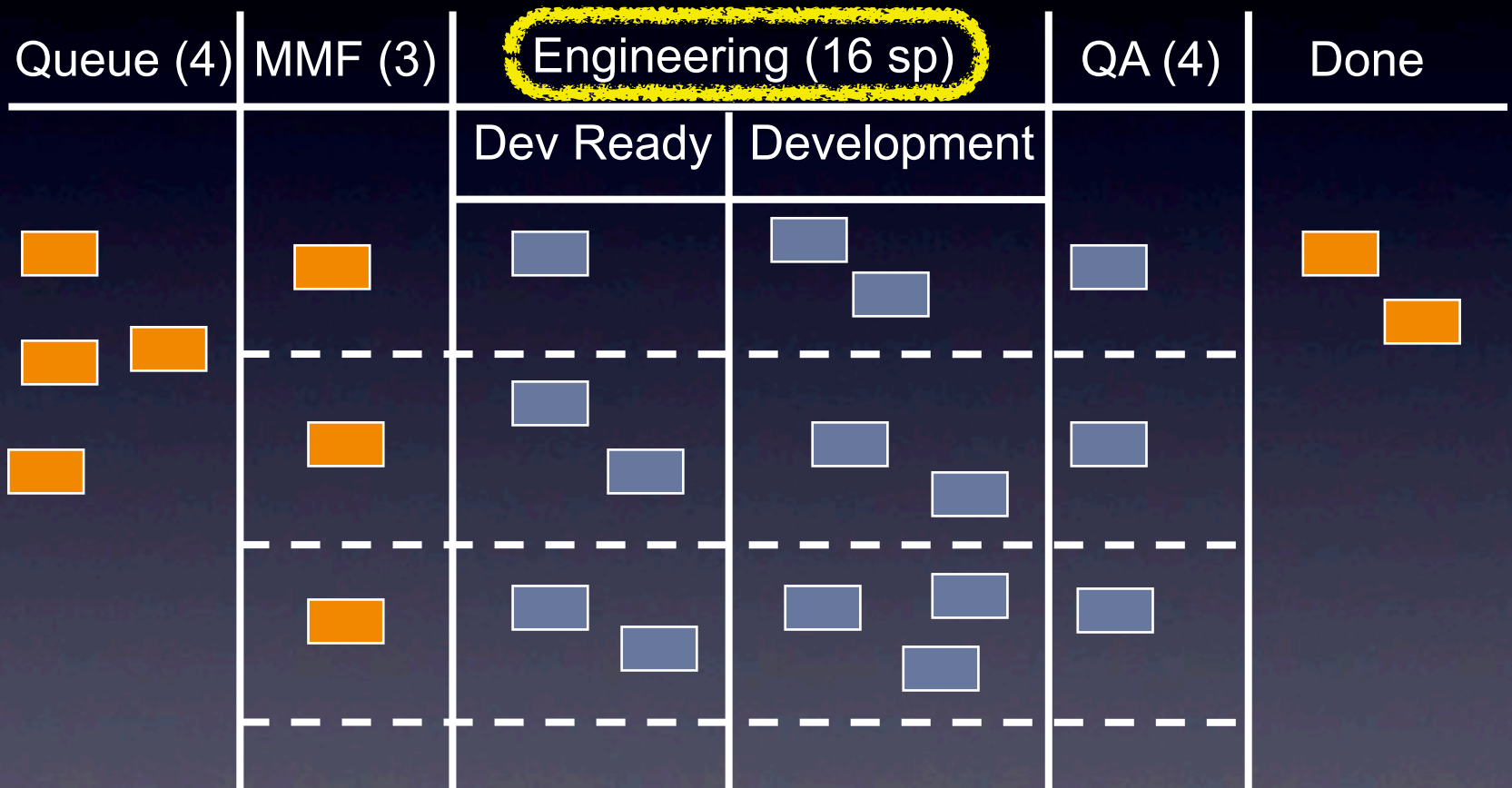
Queues and Limits

[3]

Queues and Limits

[3]

Limit By Story Points



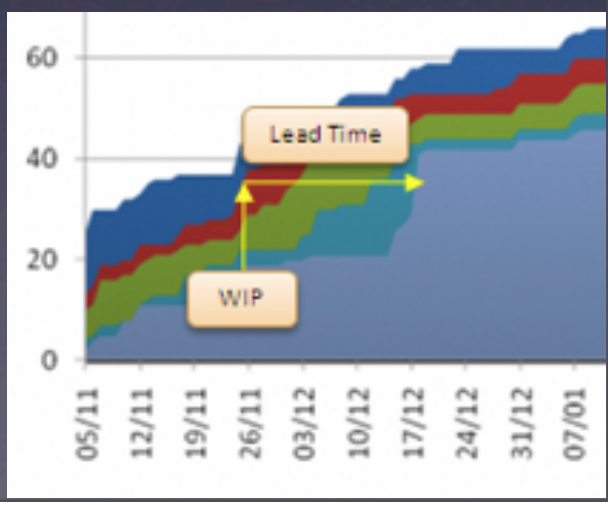
Leading Indicators

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**.

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

Its important to report on quantity and age of WIP.



[2]

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**.

With the regulating **power** of **limits**, it tells you about problems in your process, ***while you are experiencing the problem!***

[2]

Before Your Eyes

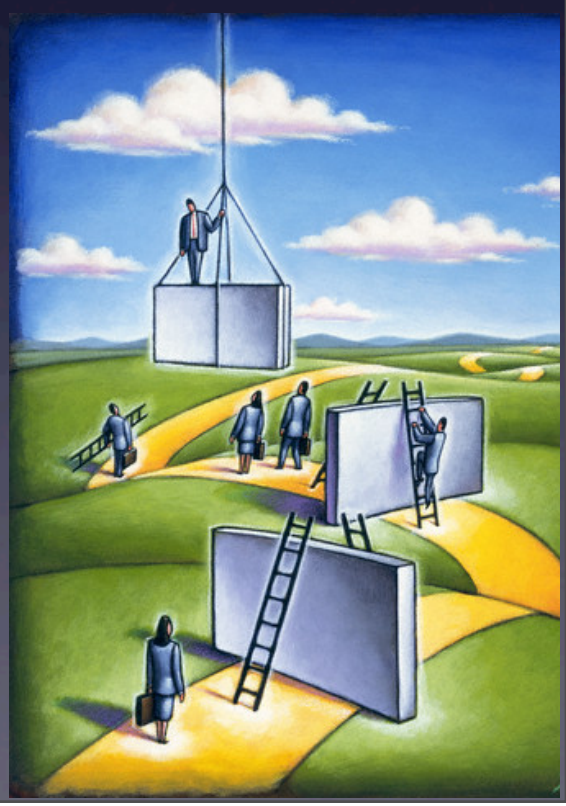
Queues start **backing up** immediately following any **blockage**.

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.



[2]

Waste Identification and Elimination

The **7 Wastes** in **Lean** apply more to **manufacturing**.

The **"waste" metaphor** is not proving 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**

[1]

Cost of Delay

Cost of delay (waste) comes in **three** abstract types in software engineering

- **Rework**
  - Defects, failure demand
- **Transaction costs**
  - Planning, releasing, training
- **Co-ordination Costs**
  - Scheduling, resources

[1]

Managing Risk With Classes of Service

A Kanban **card** can have a class of service, an **indicator** that speaks to the **risk associated** with that **feature**.

Classes of service are typically **defined** based on **business impact** and **cost of delay**.

Delivery times and **pull priorities** will **vary** across the different classes.

Classification will result in a specific set of **service levels**, that are **unique** to a line of business.

[1]

Classes of Service

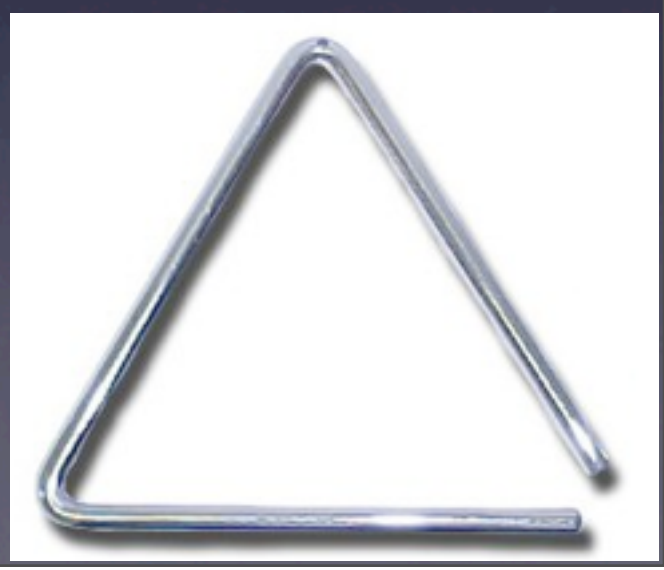
Traditional project management treats every item **homogeneously***

Kanban allows us to **break** the **triple constraint** and **optimise delivery** based on **risk**.

Example service classes:

- **Expedite**
- **Fixed Delivery Date**
- **Standard (FIFO)**

Kanban allows us to adjust to changing conditions between releases and gives us a fine-grained control on individual items.



[1] * from a risk perspective

Policies

Each service class comes with its own set of simple **policies** that affect **prioritisation decisions**.

Polices can include; **pull priority**, **limits**, **time and risk constraints**, colours and annotations.

- Fixed delivery items must enter the system based on current flow time estimate.
- Only 1 expedite request on the whole board
- At least 4 standard items (the cannon fodder that can afford to be late)
- If total WIP is 12 we want to ensure that 6 items are high priority at all times.

[1]

Classes of Service

Work **items** should **flow** through the system in a **risk optimal fashion**. The **result** should be **risk optimal releases** of software, which **maximises business value**, and **minimises cost of delay** penalties.

Empowers **anyone** to make a properly **risk aligned prioritisation decision**, in the field, on any given day, often **without** any management **intervention** or **supervision**.

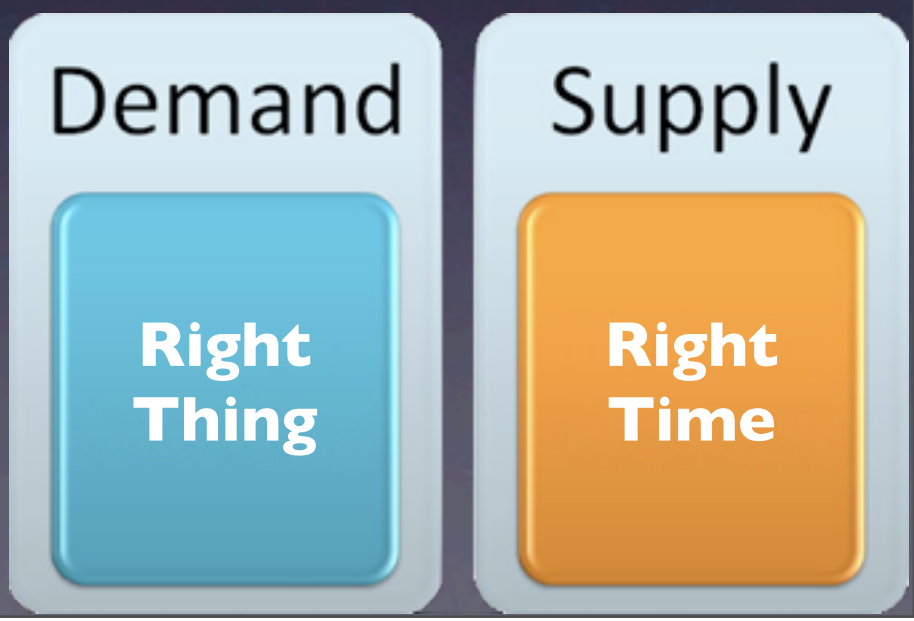
[1]

Make it Visible!

A Kanban system 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**.



[1]

Make it Visible!

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

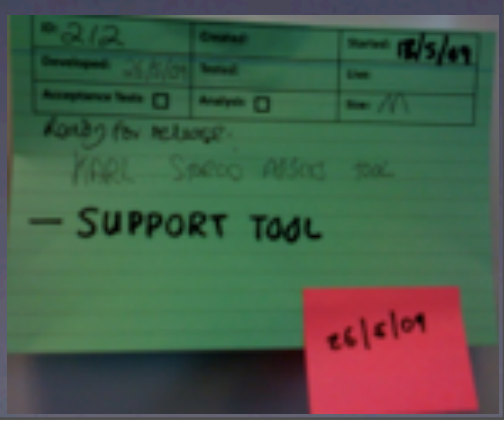
- **What** we are working on?
- If we are **overloaded**?
- Where **bottlenecks** are?
- Where **gaps** are forming?
- What is **blocked**?

Typical Agile board

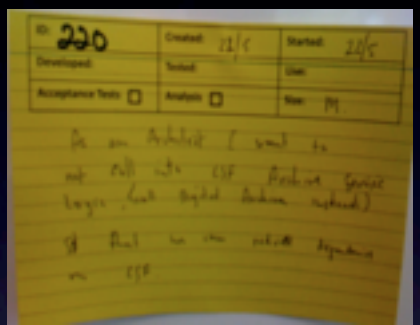
Make it Visible!

Using **colour** and **annotation** for different type of work, can quickly enable a team:

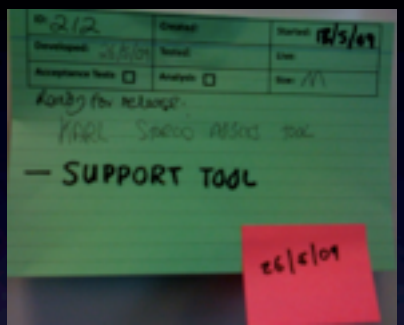
- to see what they are working on; those items giving **value** and those items **stopping** us from doing so
- **improve standups**; visual indication of blocked items
- do **simple tracking**; items passed due date, blockers
- **classes of service** and **risk** management



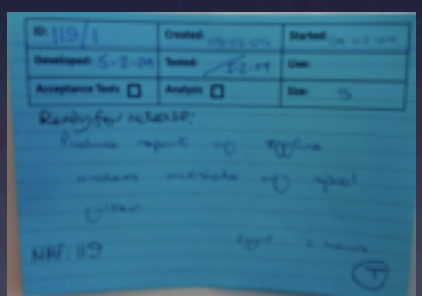
Giving Business Value



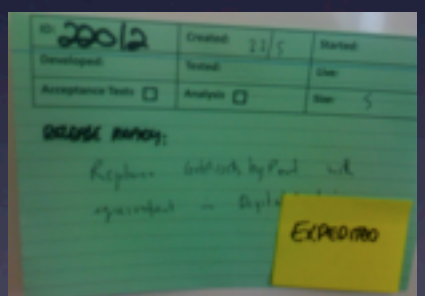
New Feature



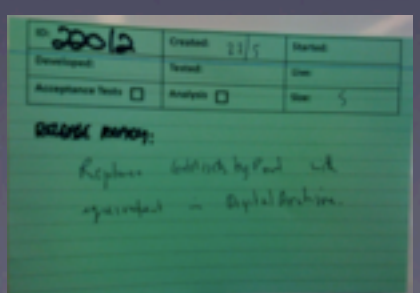
Fixed Delivery Date



Change Request

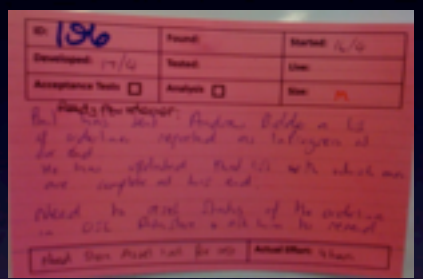


Expedited

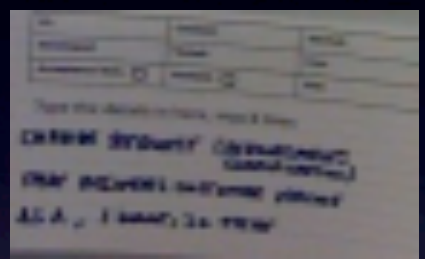


Story

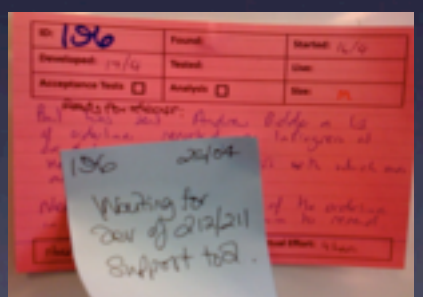
Detracting From Value



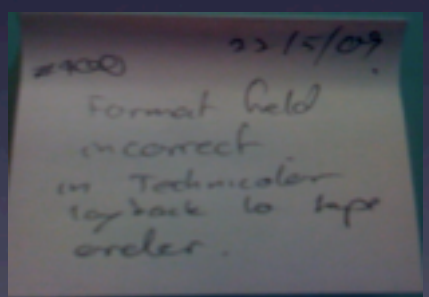
Live Defect



Missed Requirement



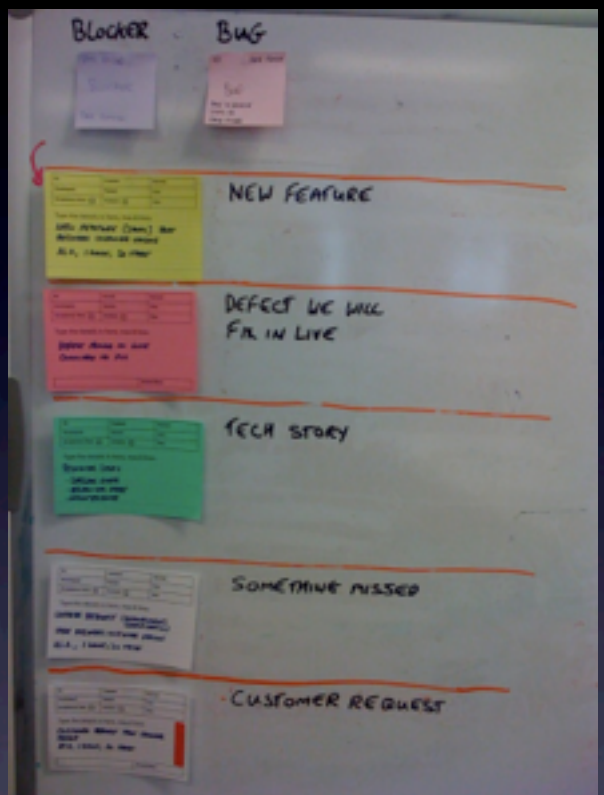
Blocker



In Process Defect

Make it Visible!

A **key** is stuck on the Kanban board.



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

ID:	Created:	Started:
Developed:	Tested:	Live:
Acceptance Tests ☐	Analysis ☐	Size:

Avatars are placed next to items in process. Helps limit work too!

Kanban Board

Does the Kanban board have to look like this?

No, its just an **example**!

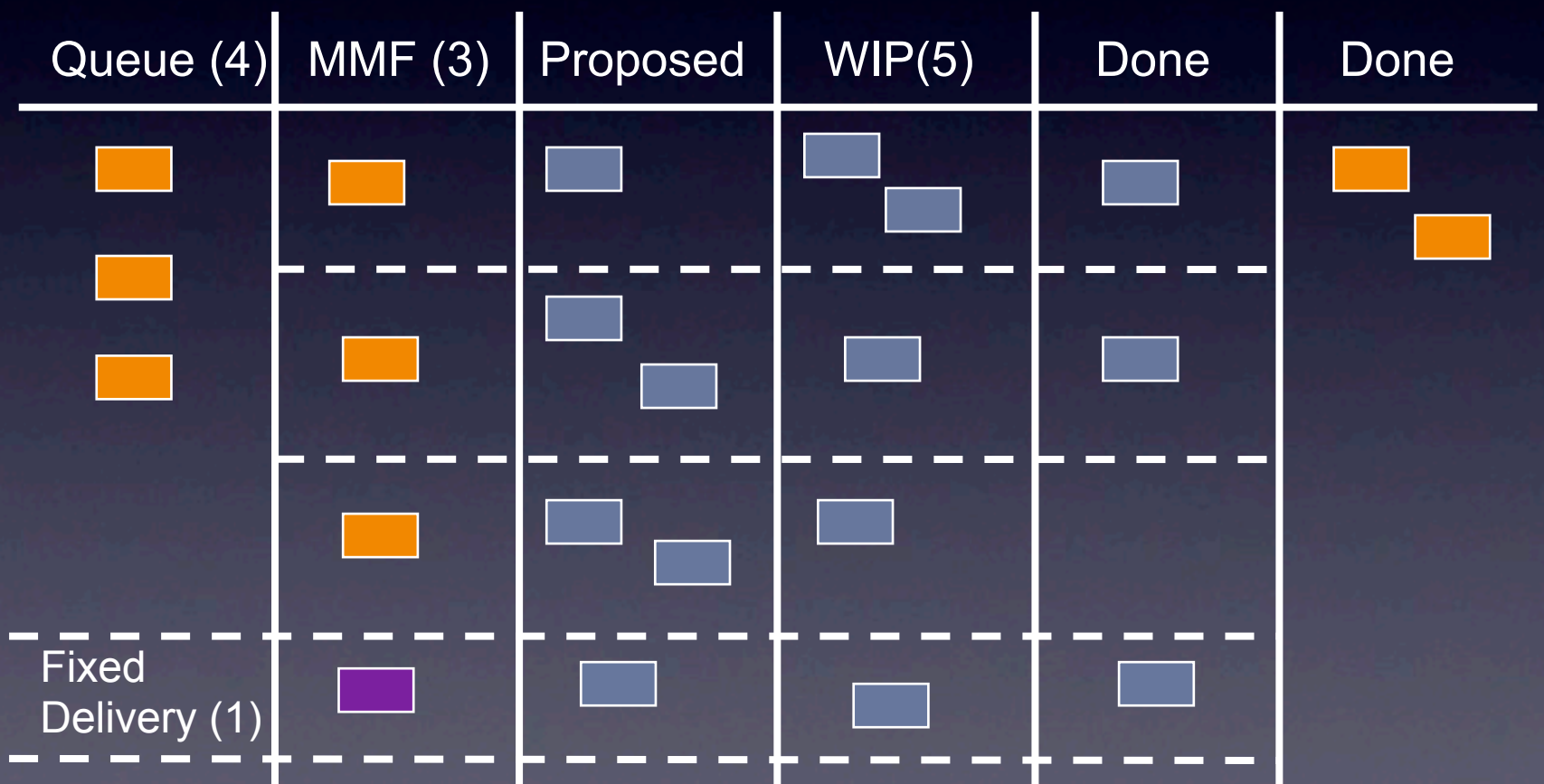
Start **simple**, **review**, and **modify** as necessary.

[7]

Two Tier Kanban Swimlanes

Swimlanes can be used for work item **types** or **classes of service**.

Explicit WIP **limits** can be set on the **swimlanes**.



[3]

Two Tier Kanban

Making Lean Values Actionable

A **Lean decision filter helps** us make **decisions** around applying Lean practices:

1. **Value trumps flow**

   Expedite at the expense of flow to maximise value

2. **Flow trumps waste elimination**

   Increase WIP, if required to maintain flow, even though it may add waste

3. **Eliminate waste to improve efficiency**

[1]

What Do I Work On Next?

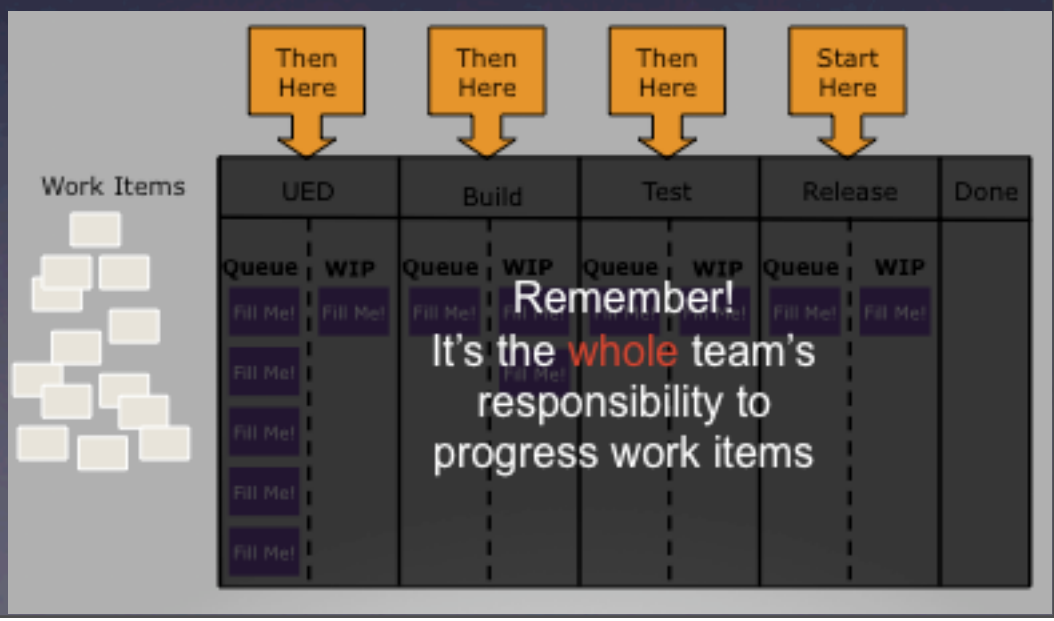
Its **easier** to start **new** work than it is to **finish** something
"I'm waiting on x"

1. Can you help **progress** an **existing item**? 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.



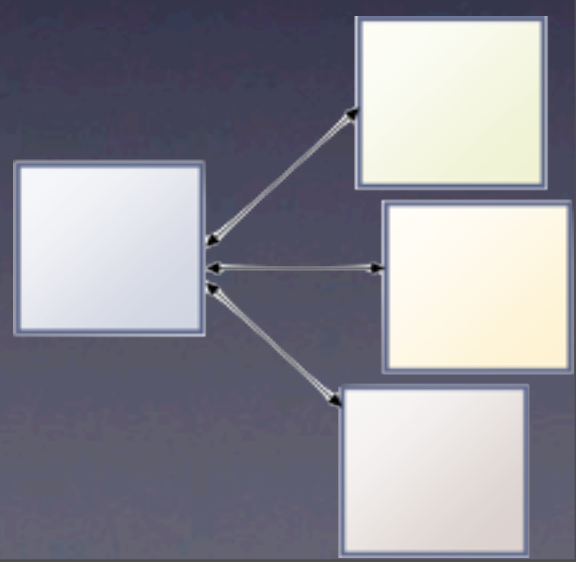
[3]

Exploding Items

After **pulling** an item from the input queue you might realise it is **too big**

2 choices

- **Kick it back** to be broken down in a **planning session**
- Break it into **smaller items now**
  - All items **reliant** on each other?
    Leave them on the board, even if it breaks the WIP limit.
    Take the **pain** now. **Value trumps flow**.
  - Some or all are **interdependent**?
    Kick some items **back** into the backlog, to be **pulled** at a **later date**.



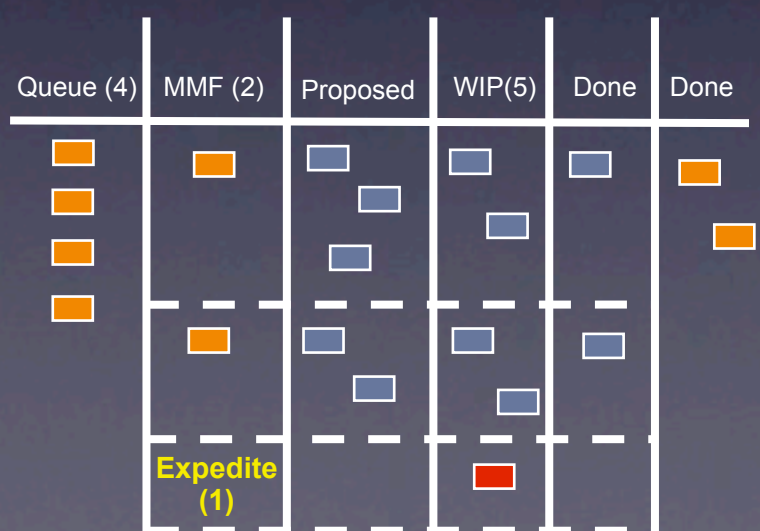
Urgent Work

If there's an **emergency**, a **support request**, a **live bug**, there's an empty slot on the board marked "**Expedite**."

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

If the "Expedite" slot is **full** and **another** urgent item appears, it has to be added to the **backlog** queue.

Slot is itself **limited**.



[4]

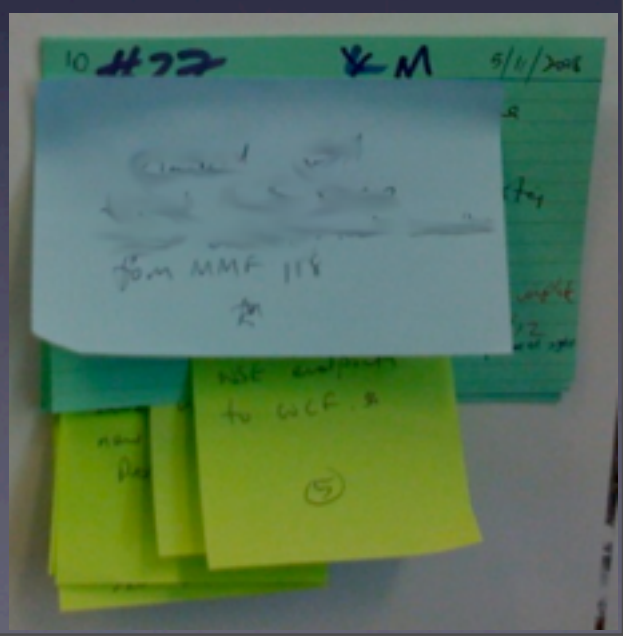
Bugs and Rework

What do we do when we **find a bug**?

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 **expedite swimlane**

Quantity of bug tickets on the board is an immediate indicator of development quality, that is impeding flow of customer valued work and reducing throughput. [1]



Blockers

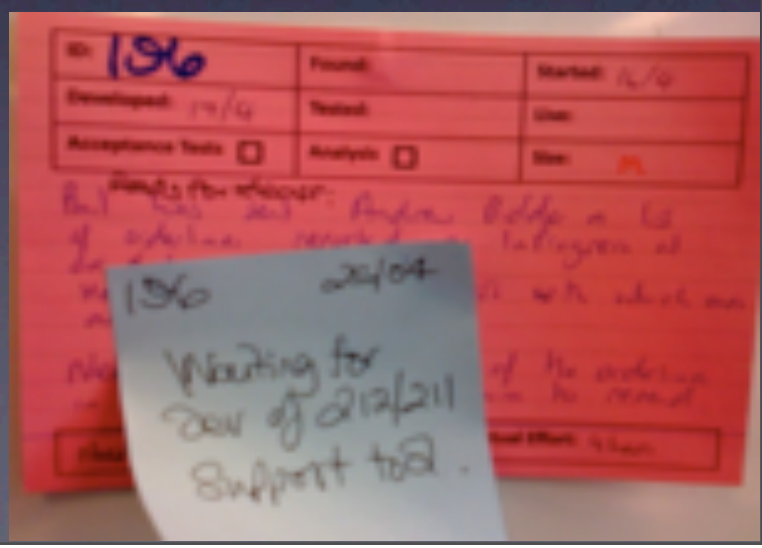
There are 3 options:

- **Leave** it blocked. **Start new work** - value trumps flow
- **Kick** it **back** to the backlog
- **Stop** the line

Share the **pain** and **learn**.

Perform **causal analysis** and **resolve**.

Quantity of blocker tickets on board directly indicates flow impacting problems, that need attention from management. [1]



Bottlenecks

There are two ways to see bottlenecks:

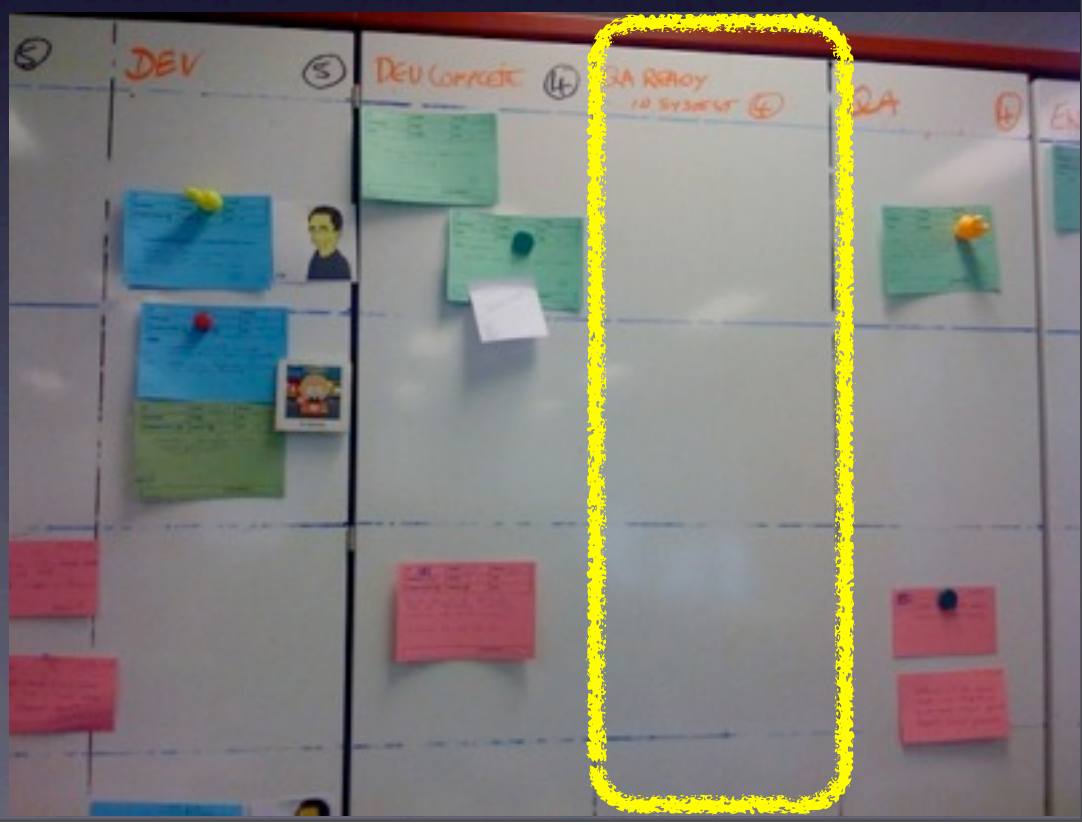
- Observe things aren't moving or **stalled**
- Observe **vacant space** on the kanban board

[1]

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.

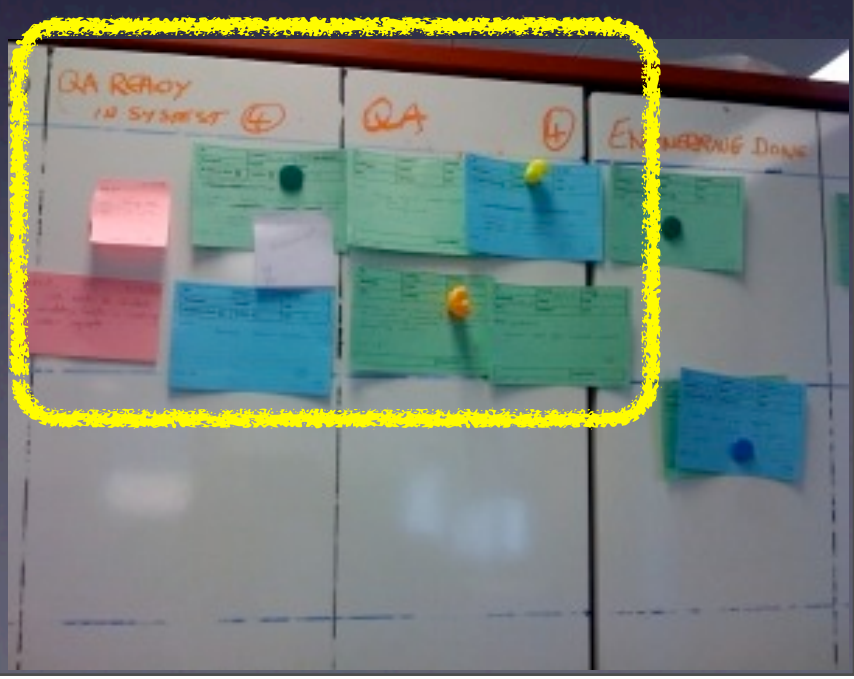


Pipeline Stall

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.



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.

[1]

Kanban Workflow

The **right work** is 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.

[2]

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.

[2]

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**.

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, the board is ignored
or it is woefully out of date,
during the standup team members
scrabble around updating it.

[2]

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.

[2]

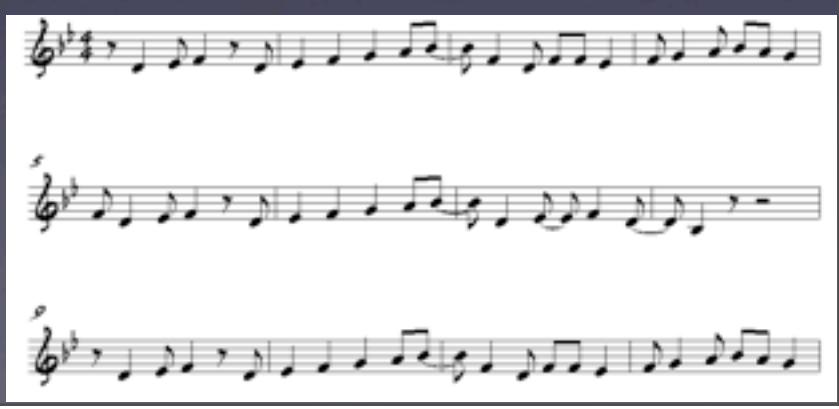
Kanban Rhythm

Standups

- Is the **board correct**?
- Do we have a **blocker** not dealt with?
- Do we have a **bottleneck**? (look for congestion or gaps in the queues)
- Are we keeping to our **WIP limits** and **classes of service**?
- Are **priorities** clear?

Post Standup

- Update charts
- Remove done items off the board



Lead Time and Cycle Time

Lead time clock starts when the **request** is made, and ends once **delivered**. What the **customer sees**.

Cycle time clock starts when **work begins** on the request, and ends when the item is **ready** for delivery.

[2]

Throughput

The **rate** of **delivery** of **customer valued** work into production.

Two major **variables** regulate Throughput; **WIP** and **Cycle Time**.

Allows **forecasting** of future **capability**.

Queuing Theory

To **reduce** 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**

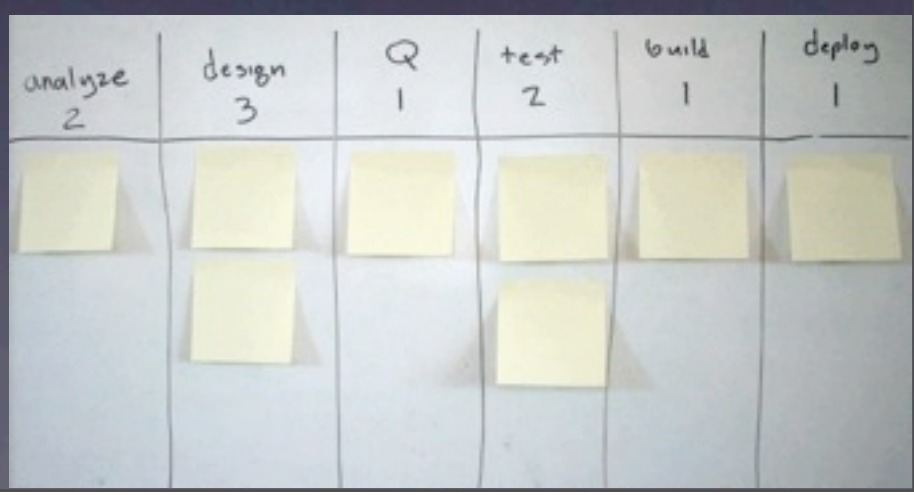
Reducing is **not** the **goal**, but is a **consequence/indicator** of **good flow**.

[1]

Flexible Control

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**



[2]

Where Is Kanban Working Well?

★ IT Application Maintenance

Examples include Microsoft, Corbis, Robert Bosch, BBC Worldwide

★ Media Sites and Applications

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

★ Games Production and Design Agencies

Where there is a lot of specialisation and a lot of hand-offs, 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 your **constraints**; whatever you have **least of** e.g. analysts or testers.

Take an **evolutionary** approach and keep changes **small** and **incremental**.

[2]

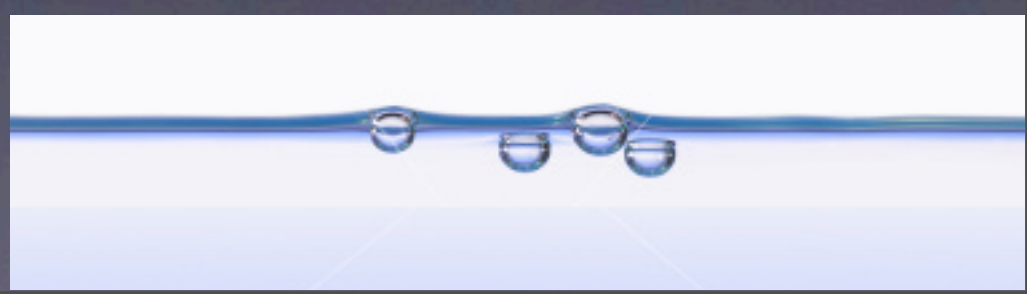
Putting It All Together

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

Based on the **live behaviour** of the workflow make hands on **adjustments** to enable **smooth** flow.

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.



[2]

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.

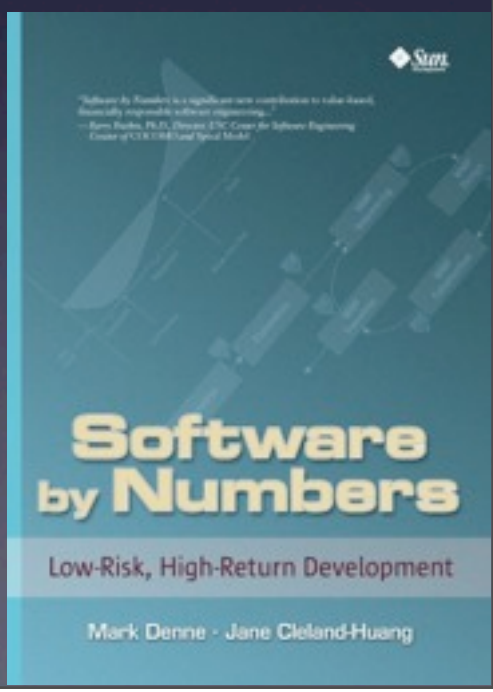
It make take a while to move enough work through the system, in order to obtain sufficient data.

[2]

MMF

"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.

The **MMF** focuses on the **businesses value**, which drives identification of the stories.

[10]

Return On Investment

Many organisations are now unwilling to tolerate **Payback Periods** of more than **a year**.

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

How is it possible to **release the capital** necessary to do software projects?

[10]

Features

Software products can be **deconstructed** into **units of value**.

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

A complex software product can deliver value even if it **isn't complete**.

[10]

Incremental Features Example

Why use Features?

Typically a Feature **creates** market **value** in the following ways:

- **Competitive differentiation**
- **Revenue generation**
- **Cost saving**
- **Brand projection**
- **Enhanced loyalty**
- **Market share**

[10]

Incremental Funding

Assembly of **units** of value creation, allows for funding to be more **granular**, and more closely **aligned** to **incremental delivery**.

This gives rise to the concept of **incremental funding**.

No longer have to obtain **large budget approval**.

[10]

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**
- **Executable Specifications**

We want to **manage** that flow of work, the **Ideation Pipeline**.

[15]

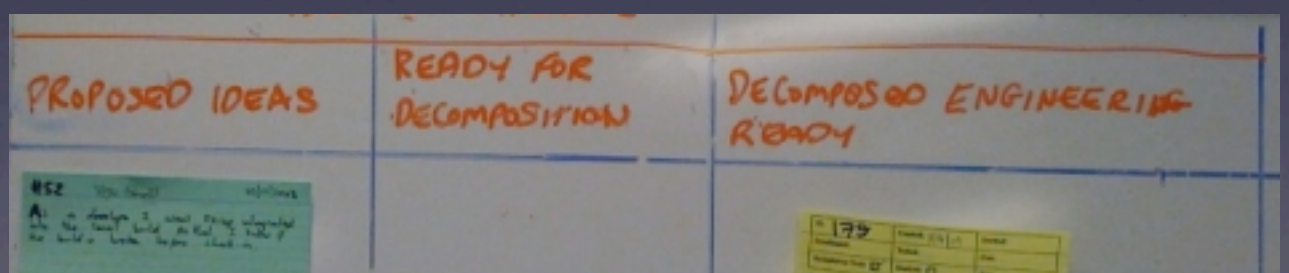
Ideation Pipeline

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** they have **capacity**.

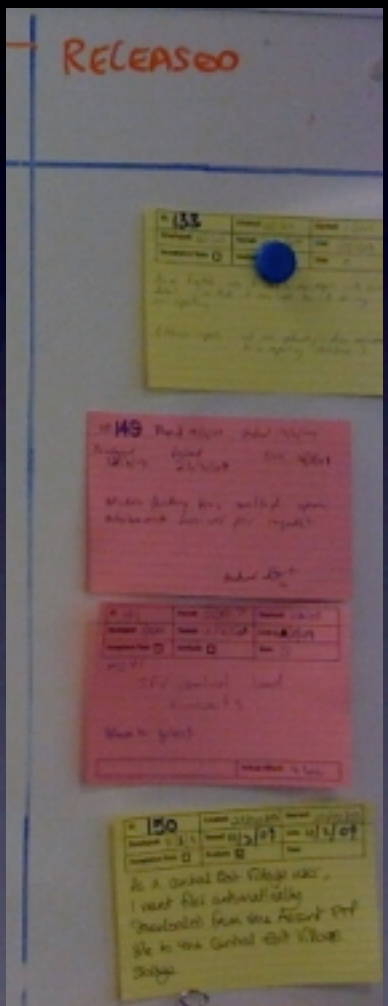
When an MMF is ready to be **pulled** *from* the ideation pipeline, the **engineering team** breaks it into **stories** and **tasks**.



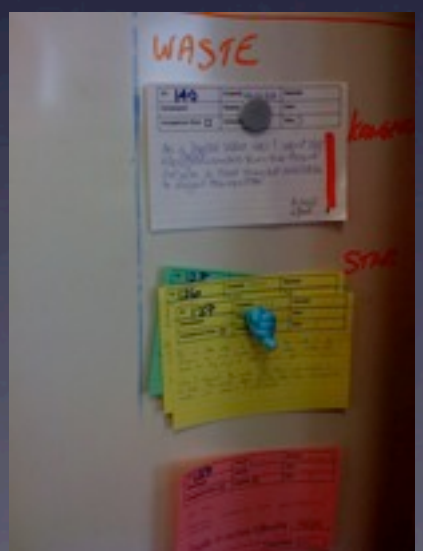
[15]

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**.



Metrics

Metrics are a tool for **everybody**

The **team** is **responsible** for its metrics

Metrics allow for **continuous improvement**

Red, **Amber**, **Green** is not enough.

Metrics

Manage **quantitatively** and **objectively** using only a few simple metrics

- **Quality**
- **Work in Process**
- **Lead / Cycle time**
- **Waste / Efficiency**
- **Throughput**

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?

Velocity can be **inaccurate** due to team **changes**.

Historical **point** valuation can **degrade** over time

Real Data

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

e.g. analysis, time to deploy, release freezes, UAT unavailability, work item type, service class.

With **real data** you can make **real commitments**.

Throughput Metrics

Total Throughput and Throughput by **T-Shirt** size.

Here we can see Throughput for the **last month**.

Last 4 Weeks

	Throughput			
Start Date	Small	Medium	Large	TOTAL
30-Mar-09	7	7	0	**14**
23-Mar-09	9	4	5	**18**
16-Mar-09	3	2	1	**6**
09-Mar-09	6	5	1	**12**

Cycle Time Metrics

Cycle Time data for our **T-shirt** sized MMFs and **per Feature Set.**

We can use this data when **estimating** upcoming work.

Mean Cycle Time - All Records

Size	Days
Small	7.4
Medium	17.0
Large	25.2

Name	Cycle	Lead
[illegible]	12.0	13.0
[illegible]	41.6	112.4
[illegible]	5.0	22.0
[illegible]	14.0	18.0
[illegible]	6.0	14.0
[illegible]	10.0	13.0
[illegible]	5.5	6.0
[illegible]	9.0	32.2
[illegible]	28.0	29.0
[illegible]	6.3	11.7
[illegible]	15.0	19.0
[illegible]	3.9	8.8
[illegible]	23.5	47.5

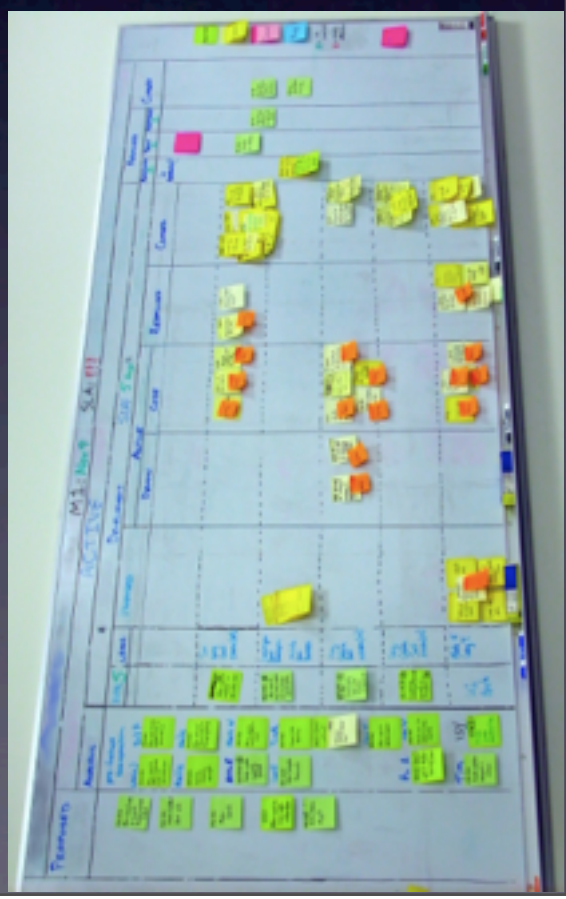
Cumulative Flow

It is like turning your board by **90 degrees** and **stacked daily**.

A great **communication** tool, showing the customer **flow of value vs time** and clear **visibility** of **events**.

Which states you track is up to you.

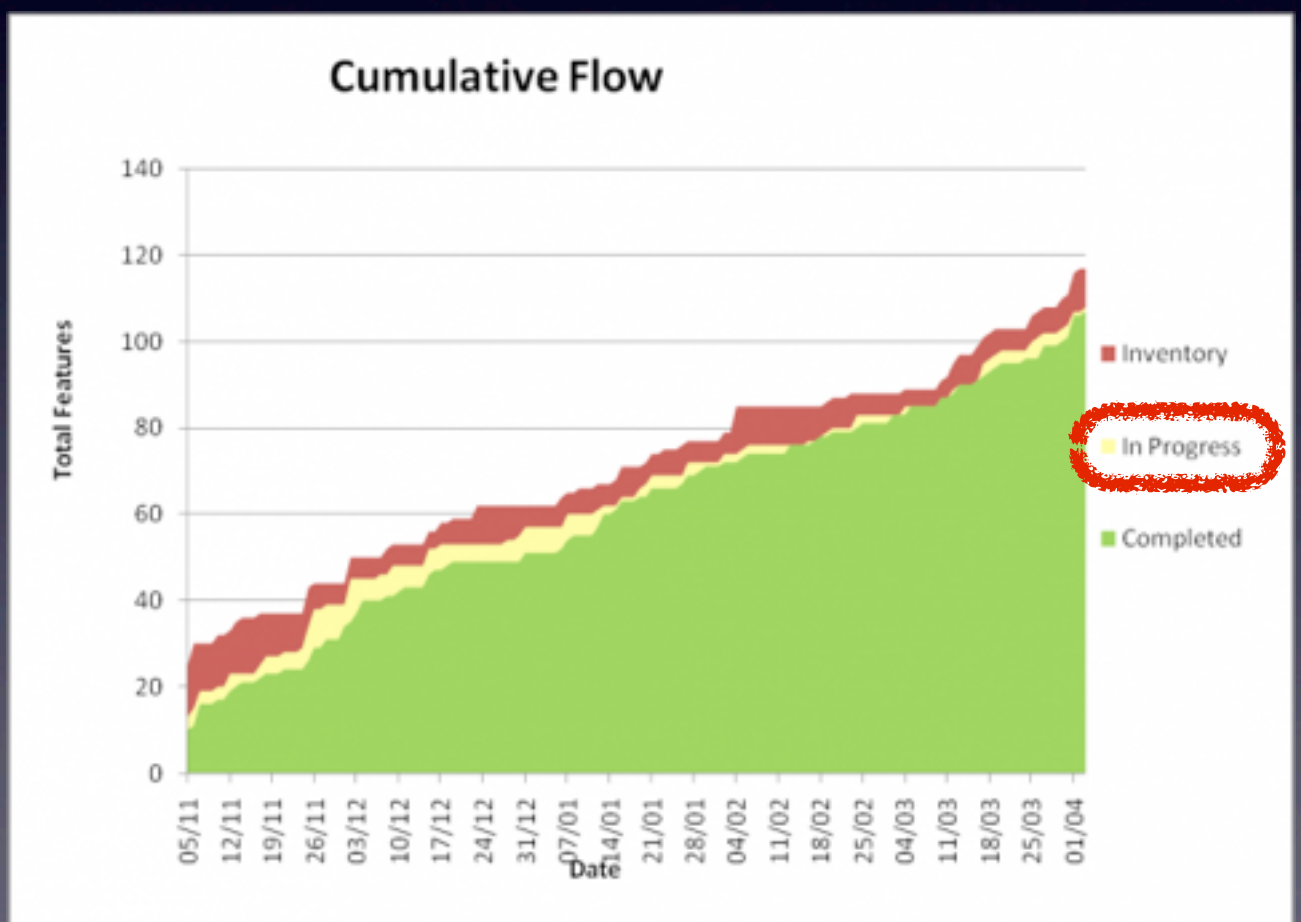
We can easily view bottlenecks using a CFD.



Cumulative Flow

This is the typical "Not Done" - "Doing" - "Done" style.

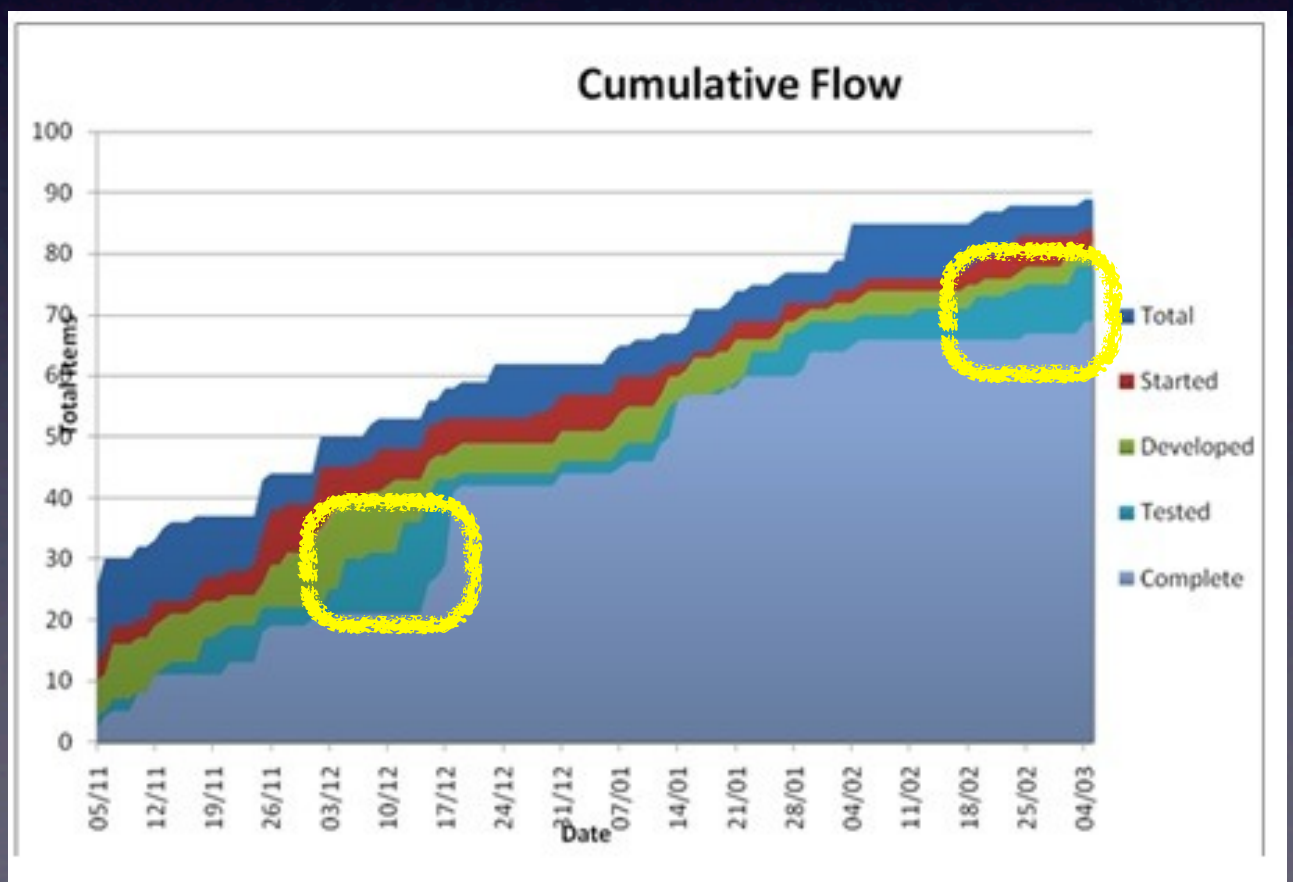
We aim to narrow the "Doing" gap using WIP limits.



More Detail Helps

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.



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**.

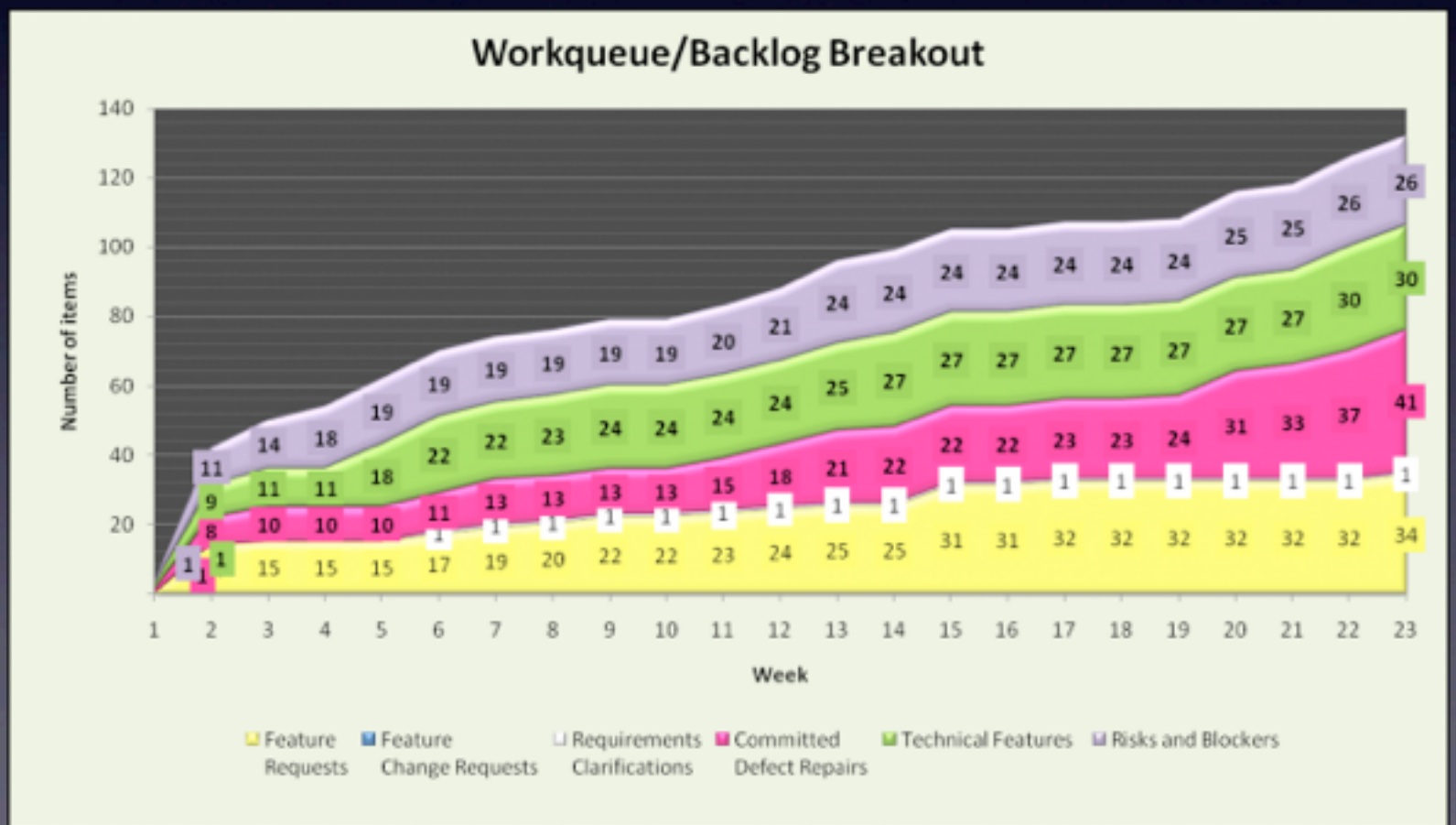
Week	Week Date	Categories that provide Business Value				Value-Categories Scope Elements
		Feature Requests		Feature Change Requests		
		New in this Week	Cumulative Total	New in this week	Cumulative Total	Cumulative Total
1	10 November 2008	26	26	0		26
2	17 November 2008	0	26	0		26

Categories that are Pure Overhead						
Requirements Clarifications		Committed Defect Repairs		Technical Stories		Risks and Blockers
New in this Week	Cumulative	New in this Week	Cumulative	New in this Week	Cumulative	New in this Week
1	1	11	11	15	15	11
0	1	1	12	1	16	1

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.

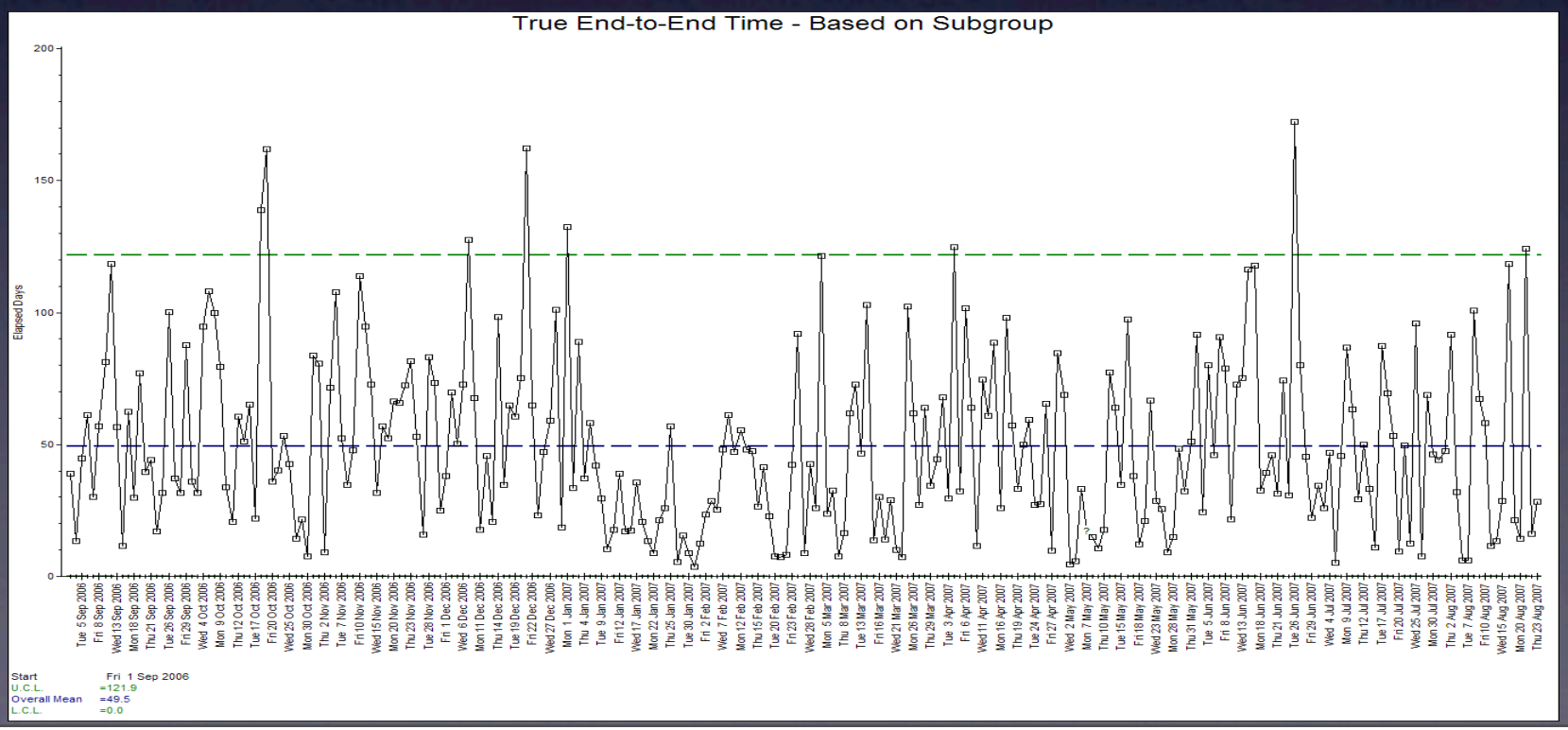
Completed component stories against total is shown.

Feature Completion			
Name	Started	Completed	%
[redacted]	1	1	100.0%
[redacted]	5	0	0.0%
[redacted]	2	2	100.0%
[redacted]	0	0	0.0%
[redacted]	1	1	100.0%
[redacted]	2	2	100.0%
[redacted]	9	8	88.9%
[redacted]	1	1	100.0%
[redacted]	3	3	100.0%
[redacted]	1	1	100.0%
[redacted]	56	54	96.4%
[redacted]	1	1	100.0%
[redacted]	2	0	0.0%
[redacted]	24	24	100.0%

Control Charts

Upper Control Limit helps **highlight** which items took **excessively long** and are likely to be **due** to **special causes** that are worth **root cause** analysing.

The **timeline view** of the control chart shows if things are **changing over time** as well as more **clearly illustrating** the **outliers**.



New Approach

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

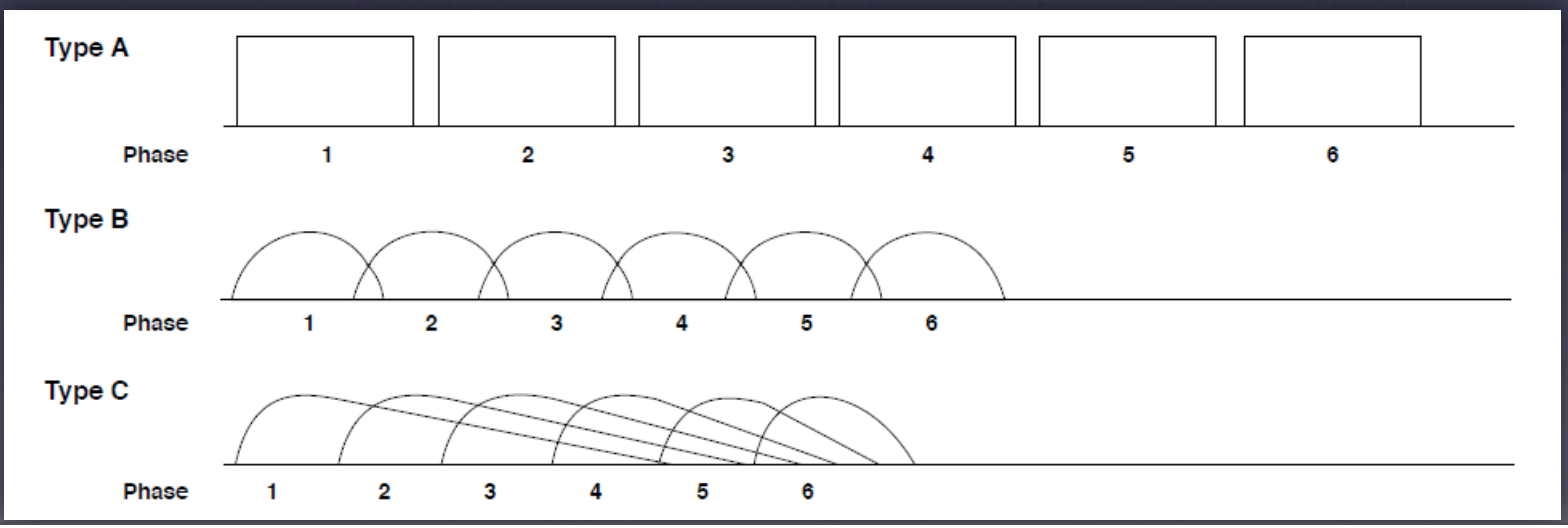
Can we go **further**?

Scrum

Origin 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 timeboxed**.

Type C is akin to **Kanban**



[12]

Involving the Business

Lean and Kanban works better, in part, because it **involves** the **business** from the **beginning**.

It does not treat the business as "**chickens**" who **have limited say** or by **proxying** them.

Kanban **breaks down** the "**them vs us**" barrier in a way that earlier agile methods have not.

Without partnership we risk just **building** the **wrong thing right**.

[1]

Flexibility

- **Agreement** on **how** the **system works** is made.
- Business understand the **policies**, and how **changing** them **affects** things
- Classes of service provide **flexibility** to **respond**
- **Policies** around **service classes allow WIP** limits to be **exceeded** or **items** to be **shelved** or **dropped** in **preference** to another
- Most of the work is un-prioritised until it is pulled into the input queue

[1]

A New Kind of Planning

No **customer** ever comes asking for "**4 weeks worth of code**".

They want **features** and **problems solved**.

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

MMFs directly solve the **problem** that **timeboxes** merely **work around**.

Planning can be '**de-coupled**'.

[2]

De-coupled Planning

Rather than planning a **batch** of product increment, we can plan a **single Feature** at a time.

We **limit WIP** in order to **minimise** the **cycle time** of Features.

A **commitment** and **deadline** can then be made per Feature.

Striking a Different Bargain

The **quality of service** promise is:

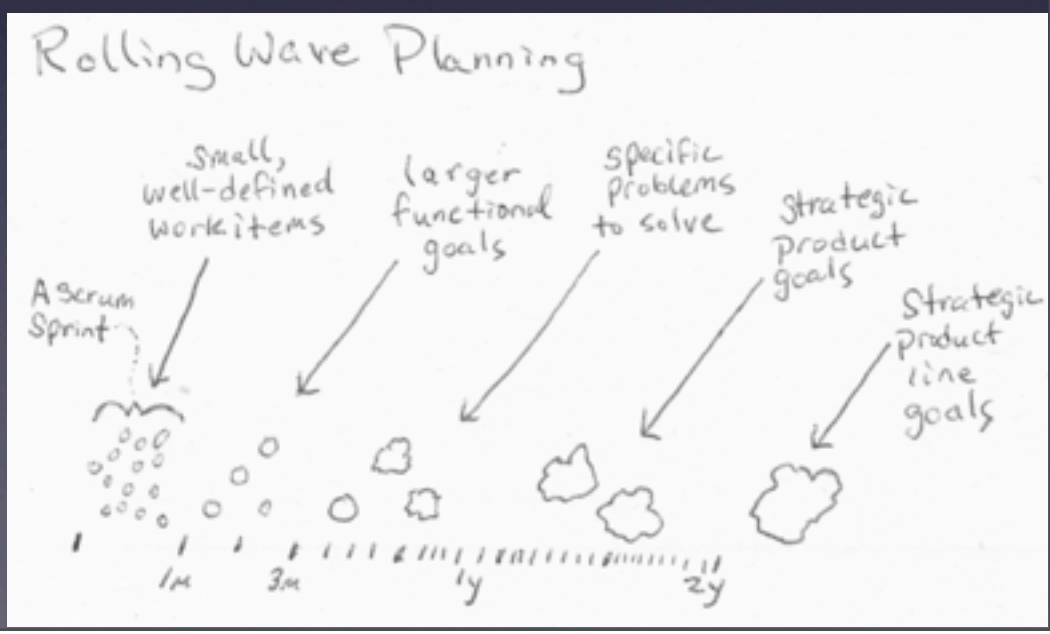
When we take on a work request, based on its **class of service**, we **intend** to deliver it **within x days**.

Not a contractual **obligation**; just a strong **indication** of turn around time.

Rolling Wave Planning

Different planning buckets for different time horizons:

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

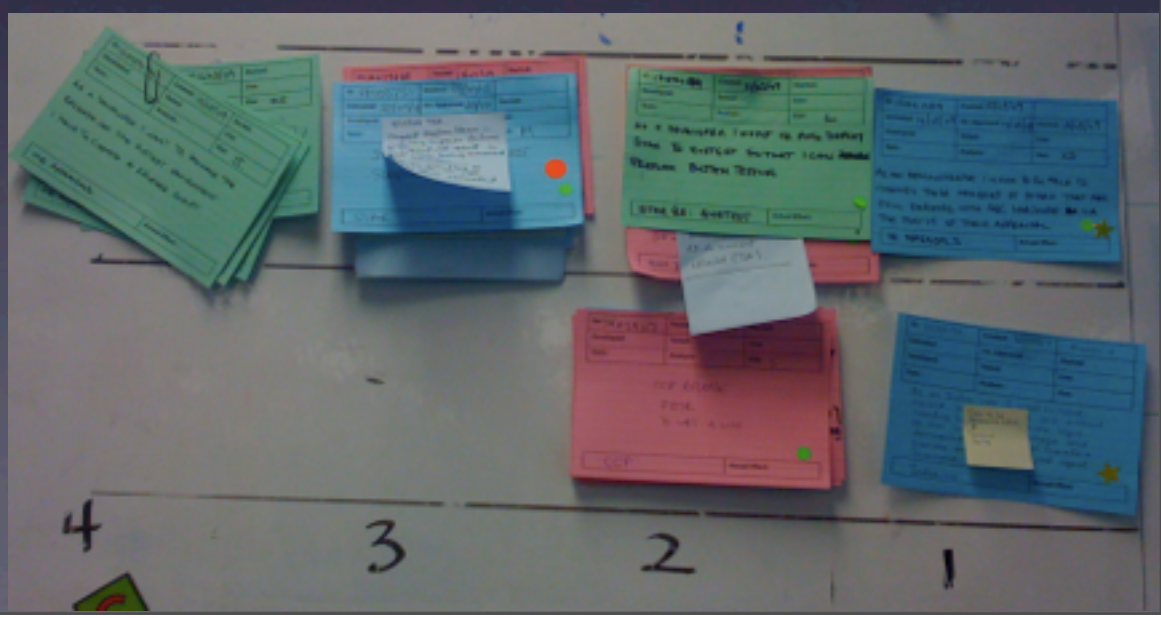


Planning on Demand

Agility implies an **ability** to **respond** to **demand**.

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

The Kanban **input queue** should reflect the **current** understanding of business **circumstances** and be **limited** *



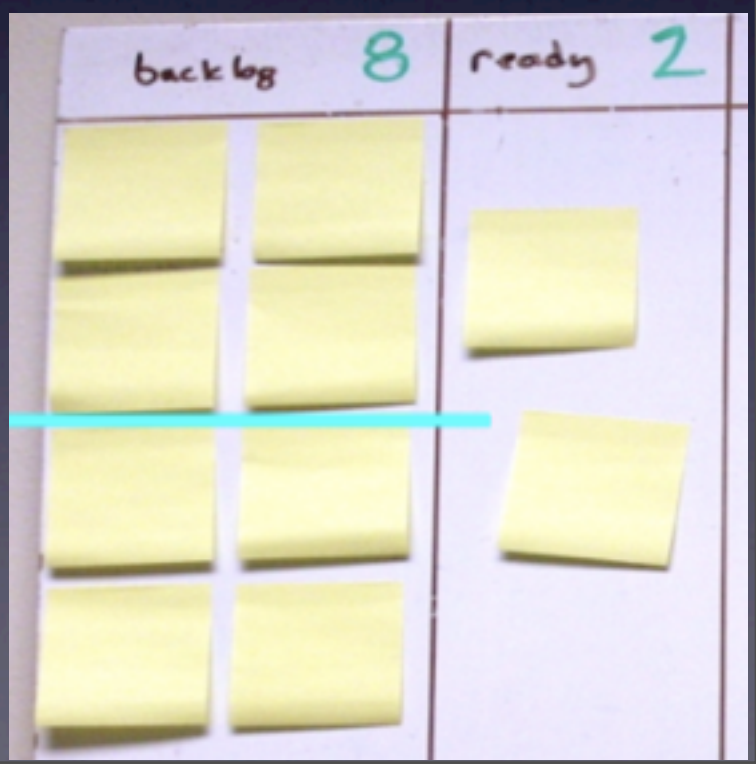
[2] *You can still have a total backlog, see ideation pipeline

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.



[2]

Releasing

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

The ideal is **deployment** on **demand**.

Releasing can be '**de-coupled**'.

[2]

Iterative Development Without Iterations

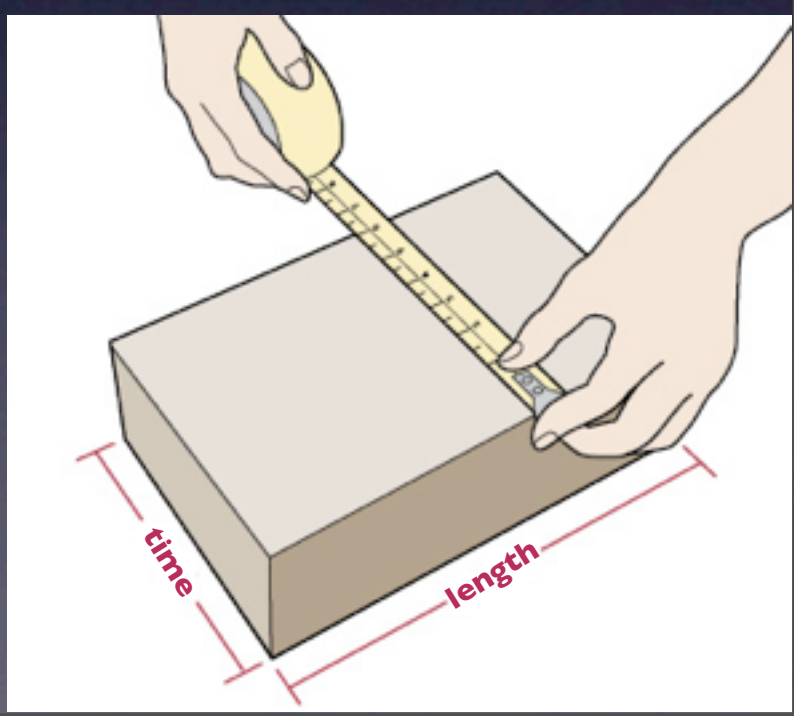
Whether **fixed length timeboxes** are useful is up to your team.

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

Nothing in the **Agile manifesto** says you have to use **iterations**.

However **nothing** in **Kanban** says you have to **drop** them.

Use **whatever works for you!**



Kanban Retrospectives

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

- Regularly **scheduled** Retrospective
- After a **feature** has been **deployed**
- When we have to "**Stop the line**"
- **Weekly mini** retrospective
- **Operations** reviews using **real data**

[1]

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**.

[1]

De-coupling

Concept that **input cadence**, **output cadence** and **cycle time** should be **synchronous** e.g. 2 week iterations, will likely to be seen as **edge case** 5 years from now.

Kanban still **allows** for **iterations** but **de-couples prioritisation**, **delivery** and **cycle time** to **vary** naturally according to the domain and its intrinsic costs.

[1]

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**.

Its 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 **estimation**
- Level 3 - **de-couple** the **release** and **planning** periods
- Finally **pull prioritised** work **on demand**

[2]

Recipe for Success

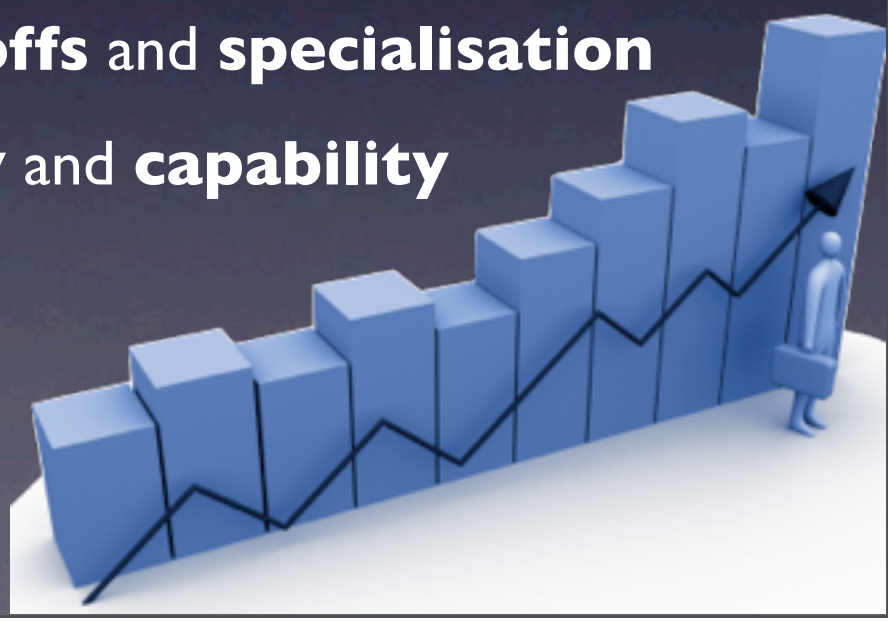
- **Focus on Quality**
- **Reduce WIP, Deliver Often**
- **Balance Demand against Throughput**
- **Prioritise**

[1]

Beyond Scrum

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**
- Involving customers on **ROI**, **capacity** and **capability**



In Conclusion

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

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

[1]

In Conclusion

Kanban **pull**, **visual control**, and **flow paradigm** encourage us to:

- **Identify** and **resolve bottlenecks**; releasing greater **throughput**
- **Causal analysis** on impediments **blocking flow**; improving **cycle time**
- **Identify** and **eliminate waste**; improving **flow** of **value**
- Encourage a **whole system** view; rather than a **locally optimised** view

Kanban (pull) changes the underlying paradigm from project-centric, to flow and value-stream centric.

[1]

In Conclusion

Kanban **allows for** workforce **specialisation**, while **exposing** the **effects** that too many **specialists introduce**, such as **bottlenecks**, **queues between hand offs**, and, consequently, **greater WIP**.

Challenge ourselves to

- **Enhance business agility**
- **Shorten cycle times**
- **Deliver more value earlier**

[1]

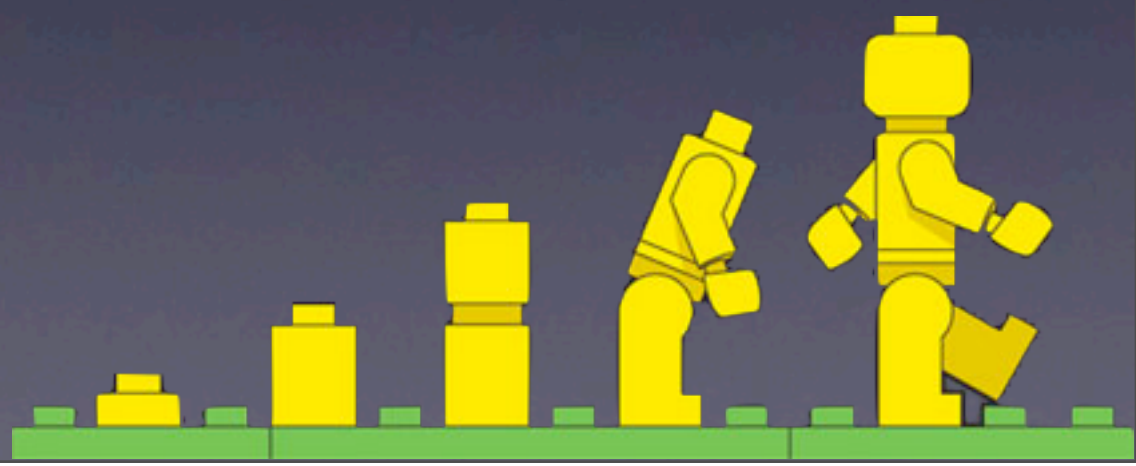
In Conclusion

Enables an **evolutionary approach** to **agile transition**.

Delivers **evolution, based** on **Lean principles** rather than **revolution** based on the **Agile Manifesto**.

Proven **easy** to **adopt** and **lowers** the **resistance** to **change**.

The result is a gradual, incremental approach to change that is empowering for everyone.



[1]

In Conclusion

You may find once you start, you begin 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!

Mix and match as you need! I can't imagine a successful Scrum team that doesn't include most elements of XP.

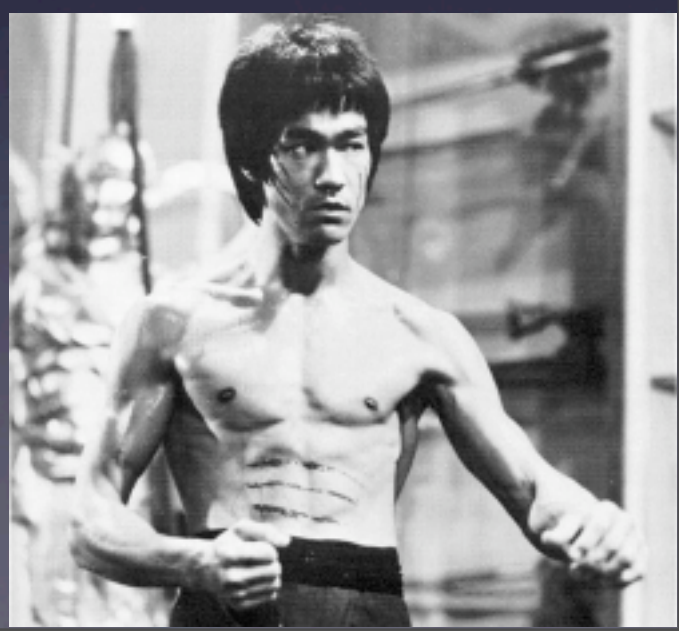
Don't **limit yourself** to one tool!

In Conclusion

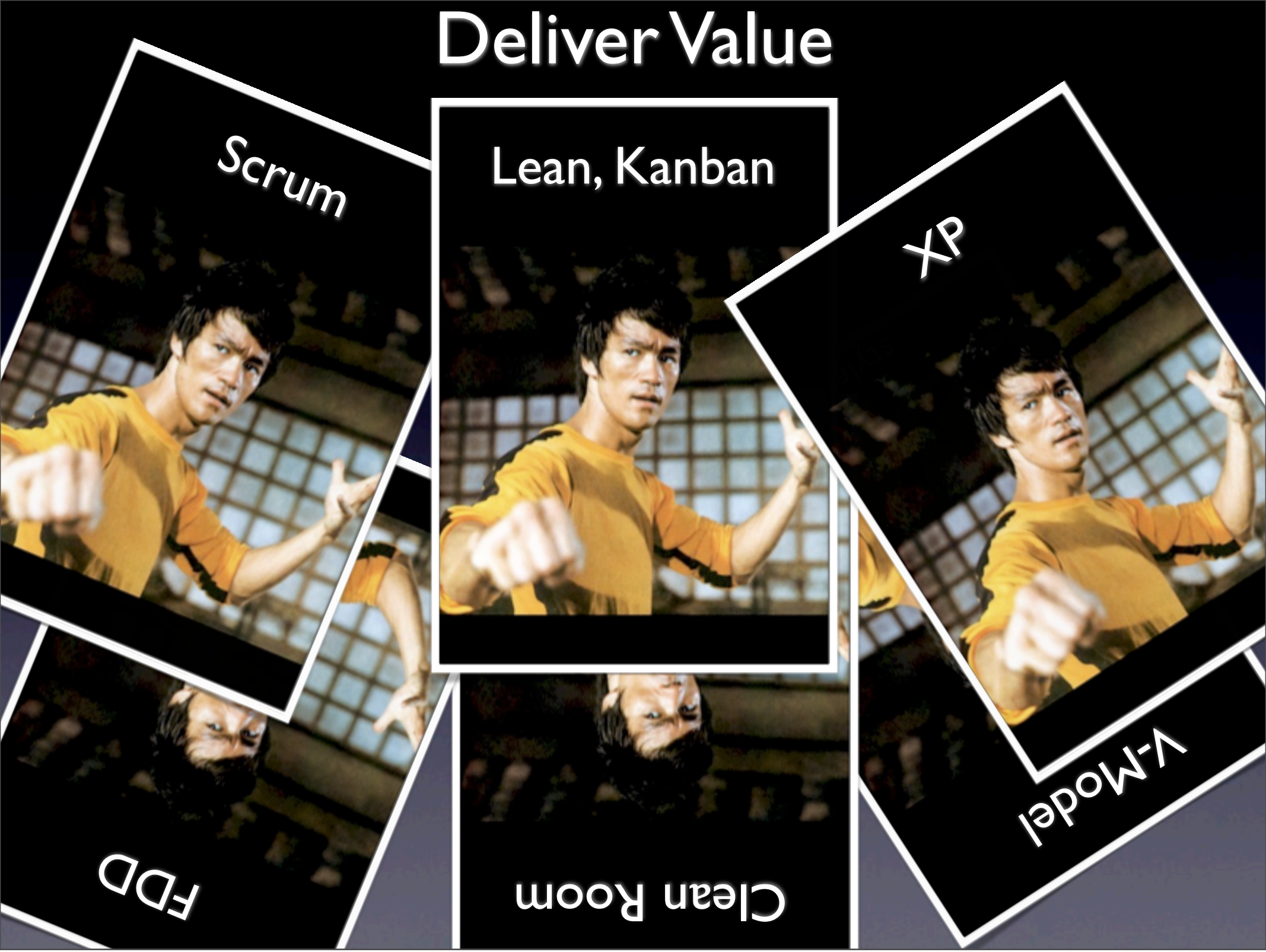
After many years of studying martial arts, Bruce Lee came to learn, that traditional techniques were too rigid and formal, to be practical.

Rather than dogmatically using one style, he taught his students to constantly adapt, and use different techniques to suit their individuals strengths and needs.

> “Absorb what is useful; disregard that which is useless.
>
> Learn the principle, abide by the principle, and dissolve the principle
>
> I hope to free my followers from clinging to styles, patterns, or moulds.”



Deliver Value



More Information

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

Kanban community site http://www.limitedwipsociety.org

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

Corey Ladas Scrumban book and blog http://leansoftwareengineering.com

Karl Scotland 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

Understanding variety of demand http://www.triarchypress.co.uk/pages/articles/Understanding-the-Variety-of-Demand.pdf

Cultural Change is Free - John Seddon http://www.vimeo.com/4670102

Suggested Reading

1) Lean-thinking, Womack and Jones
2) Managing the design factory, Donald G. Reinertsen
3) Principles of Product development flow, Donald G. Reinertsen
4) Managing to Learn, John Shook
5) Creating a Lean culture, David Man
6) Scrumban, Corey Ladas
7) Kanban x, David Anderson

Thank You!

Any Questions?

References

[1] David J Anderson - Agilemanagement.net and KanbanDev Yahoo! group.

[2] Corey Ladas - Scrumban book, leansoftwareengineering.com and KanbanDev Yahoo! group.

[3] Karl Scotland - availagility.wordpress.com and KanbanDev Yahoo! group.

[4] James Shore - jamesshore.com

[5] Aaron Sanders - aaron.sanders.name and and KanbanDev Yahoo! group.

[6] Dave Nicolette - www.davenicolette.net/agile and KanbanDev Yahoo! group.

[7] Henrick Kniberg - Scrum vs Kanban and and KanbanDev Yahoo! group.

[8] Bernie Thompson - leansoftwareengineering.com

[9] Rob Bowley and Matt Wynne - Evolving from Scrum to Lean

[10] Software by Numbers by Mark Denne and Jane Clelenad-Huang

[11] Alan Shalloway - KanbanDev Yahoo! group.

[12] The New New product Development Game, Hirotaka Takeuchi and Ikujiro Nonaka

[13] Koen Van Exem - Dimensional Planning http://www.inxin.com/

[14] Erik Willeke - KanbanDev Yahoo! group.

[15] Joe Ocampo and Steve Harman - KanbanDev Yahoo! group.

Additional Info

Kanban Brand

Kanban (large K) has become a brand. It is not the kanban (small k) signalling cards used in manufacturing.

Kanban is a true pull system for software engineering.

Kanban fully implements all 5 pillars of lean.

Scrum for software is not the same as when used in rugby, we need to understand the same difference in terminology of kanban and Kanban.

Kanban Introduction

Kanban is an emerging process framework that is growing in popularity since it was first discussed at Agile 2007 in Washington D.C.

Kanban uses a card wall combined with strict work-in-process limits to implement a self-organising pull system for project management and team organisation.

Kanban has become popular for IT Application maintenance, and for projects in fast moving domains like media and entertainment, and in domains with a lot of specialisation such as games.

Kanban Introduction

Kanban has shown a remarkable ability to encourage collaboration with other parts of the organisation such as marketing (upstream) and operations (downstream).

Kanban has enabled a Kaizen (continuous improvement) culture in such organisations and greatly enhanced the level of trust between all teams involved in delivering valuable working software regularly.

Kanban

Anyone that has used 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!

[2]

It's Not Just a Task Board

While the word Kanban comes from the Japanese for "visual card", the term Kanban as used by the Kanban Software Development community, represents much more than a standard task-board.

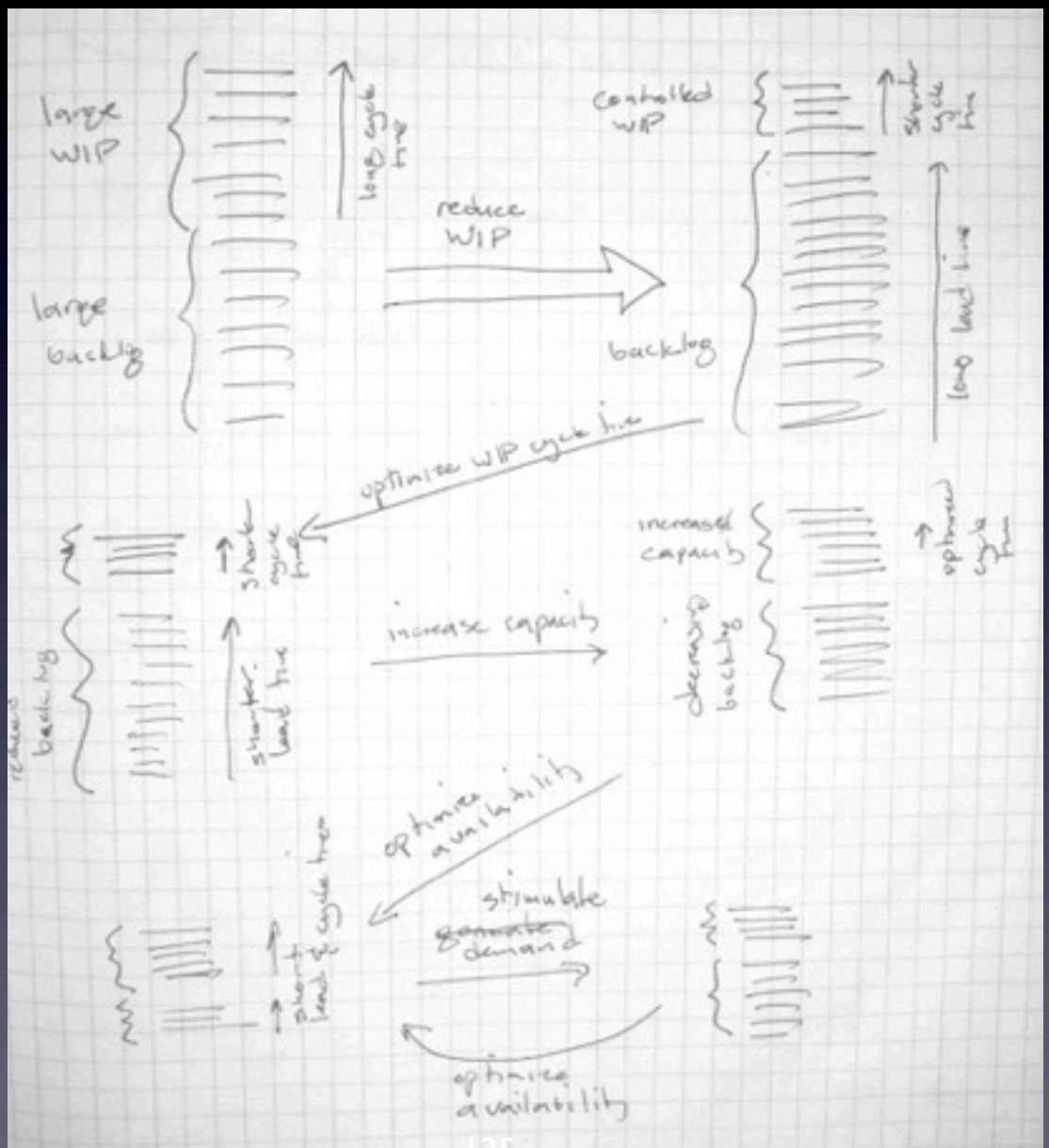
Additionally, the Kanban Software Development community have not tried to replicate the mechanism of the Toyota Production System kanban tool exactly, but have taken the underlying principles in order to achieve similar effects in software development

We can describe a Kanban System for Software Development as one which allows value to flow through the whole system using WIP limits to create a sustainable pipeline of work. Further, the WIP Limits provide a mechanism for the Kanban System to demonstrate when there is capacity for new work to be added, thereby creating a Pull System. Finally, the WIP Limits can be adjusted and their effect measured as the Kanban System is continuously improved.

A task-board simply shows what development tasks have been predicted to be done in the current time-box, with their status.

[3]

Kanban Overview



[2]

Selling Kanban To Management

Do you think my team has a limit on its capacity? – very few people will argue with that.

Do you think its better for our staff to work on one thing at a time to get work done quicker? – few people will argue with that.

I suggest we establish our capacity and manage work around this.

When adopting Kanban we aren't initially changing anything, we are making things transparent and mapping our workflow.

From there we will be looking to remove/reduce things that slow down our ability to deliver customer needs into production.

Elevator Pitch

Kanban helps our team **deliver value** to the **business**,
by **promoting flow** and **reducing cycle-time**,
through **limited WIP** and a fully **transparent value pulling** system.

Kanban is a **transparent**, **work-limited**, **value pulling** system.

Kanban (pull) changes the underlying paradigm from project-centric to flow and value-stream centric. [1]

[14]

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"

[1]

Lean

"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 Poppendiek

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 rigourous and formal engineering workflow, with multiple preventive quality control steps, and planned continuous process improvement

[2]

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

[2]

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.

[1]

Kanban is not Lean vs Agile

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]

Making Lean Values Actionable

A **Lean decision filter** helps us make decisions around applying Lean practices:

1. **Value trumps flow**

   Expedite at the expense of flow to maximise value

2. **Flow trumps waste elimination**

   Increase WIP if required to maintain flow even though it may add waste

3. **Eliminate waste to improve efficiency**

   Do not pursue economy of scale

[1]

Making Agile Values Actionable

- Are we encouraging a High Trust Culture?
  - Empowerment
  - Collaboration
  - Tolerant of Failure / Encourage Innovation
- Are we treating WIP as a liability rather than an asset?
  - Reduce delivery time
- Are we making progress with imperfect information?
  - "perfect is the enemy of good enough"

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.

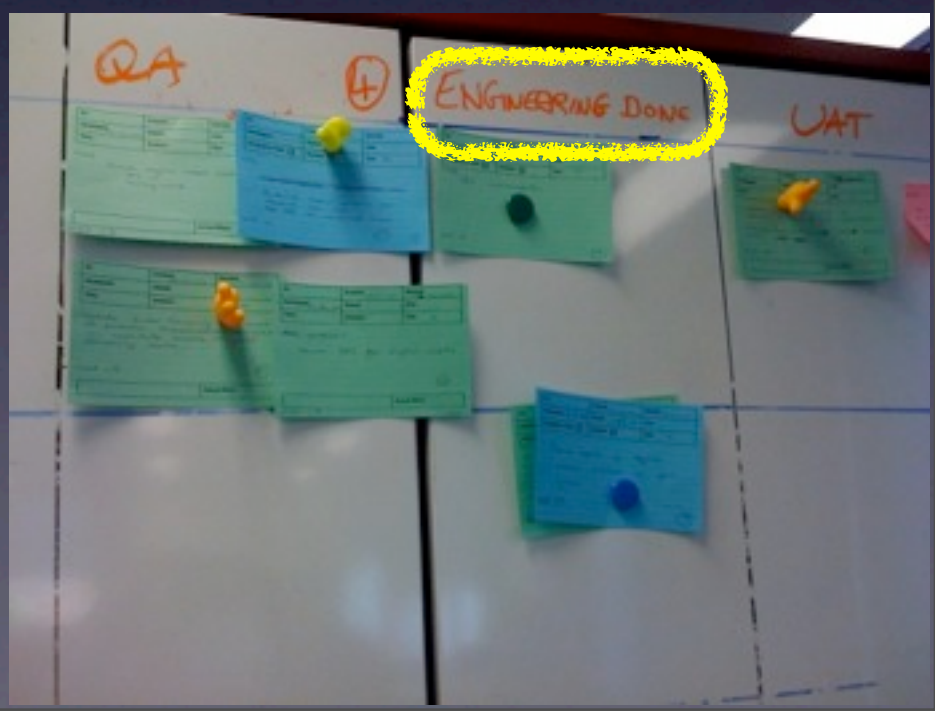
I want to prevent Kanban (in software engineering) becoming a dogmatic superstitious practice level process. What is different about Kanban is it does not specify engineering, workflow or project management practices. These are left to emerge in a situational/context specific fashion. In this respect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

[1]

Engineering Cycle Time

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



Cycle Time Metrics

Cycle Time data for our T-shirt sized MMFs.

We can use this data when estimating upcoming work.

Mean Cycle Time - All Records

Size	Days
Small	7.4
Medium	17.0
Large	25.2

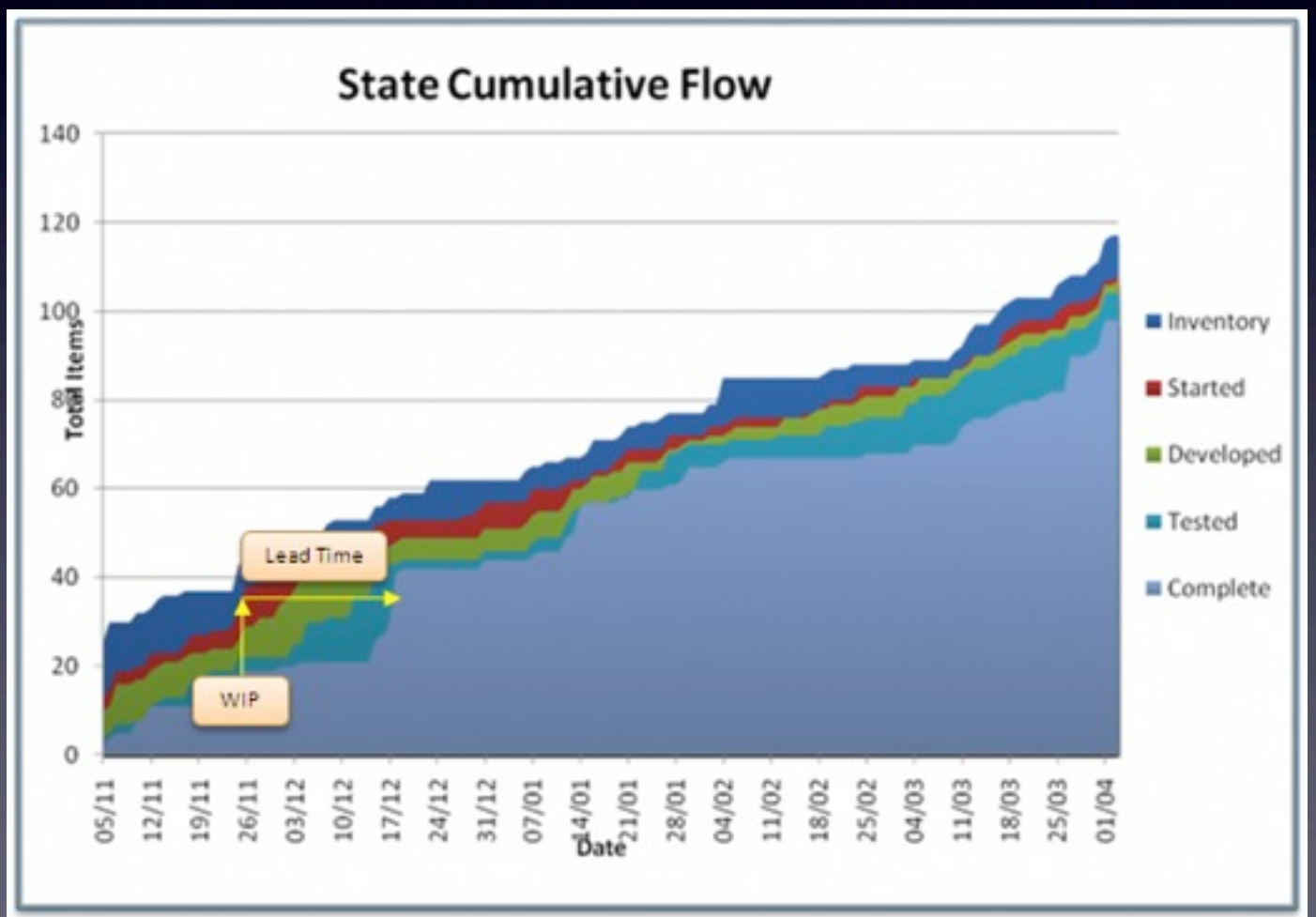
An additional 10 days to release Large items once built, something to improve!

Mean Cycle Time - Engineering

Size	Days
Small	6.4
Medium	12.5
Large	15.0

Cumulative Flow

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



Make it Visible!

- What are we **working on**?
- What is **blocked**?
- Are we **overloaded**?
- What **bottlenecks** do we have?

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.

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.*

Estimationless

Removing estimation is a choice based on risk.

When applicable replace estimation with a commitment to a regular release.

Use classes of service to mitigate risk.

Kanban does not mandate estimationless approach. Estimationless process is a risk aware choice.

Cost of Delay Examples

- **Rework -** Examples; bugs found whilst in process, building the wrong thing.
- **Transaction costs -** Examples; making a plan, work generated outside of the team, people attending a planning or release meeting and not doing their normal work for the duration of that meeting, training people on new features, out of hours work.
- **Co-ordination costs** – Examples; scheduling meetings and facilities, finding the right people and getting them together, gaining sign off of a new release, working with 3rd parties, taking a live product off line.

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.

[5]

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 they will be regularly reviewed with the Lead Developer and Architect.

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.

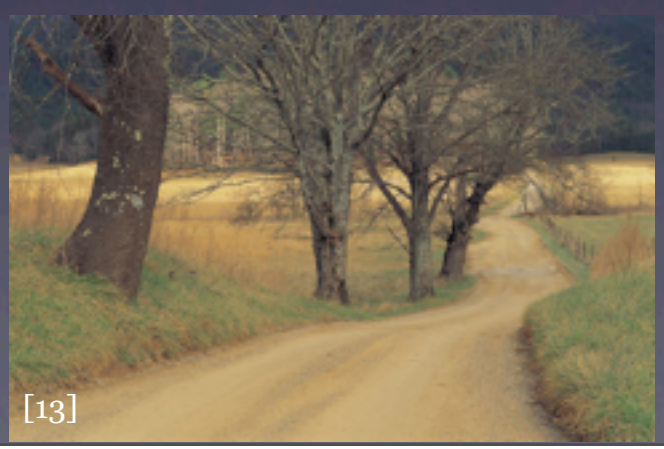
Dimensional Planning

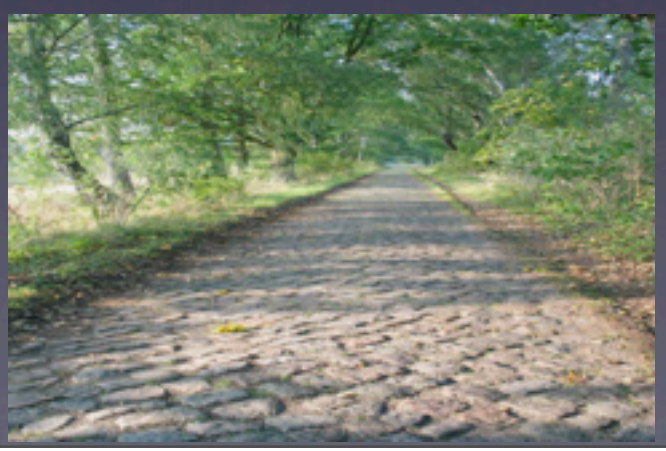
For the different depths we use the following levels:

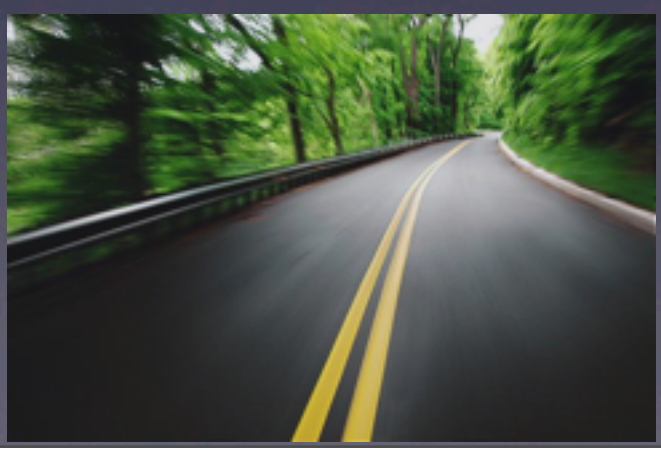
dirt road: This level is the minimum implementation with manual workarounds. Its recognised that it will have a limited life span.

cobblestone 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.


[13]





Retrospectives

Higher maturity Kanban teams challenge themselves with questions like

- How can we enhance business agility?
- How can we shorten cycle times?
- How can we deliver more value earlier?

The key is to look at the issues that are causing impediments.

It's not enough to be good at reacting and producing work-arounds or resolutions. You have to look at root cause and go and eliminate it.

For example, if sloppy requirements cause a lot of impediments that impact cycle time, then improve the requirements solicitation method.

[1]

Mini Retrospectives

- **Weekly mini** retrospectives
  - 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

You need slack to enable reflection, learning and actioning improvement opportunities. Without slack there can be no kaizen.

[1]

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.

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.

[2]

Queuing Theory

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

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

* Little's Law for Queuing Theory

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

[2]

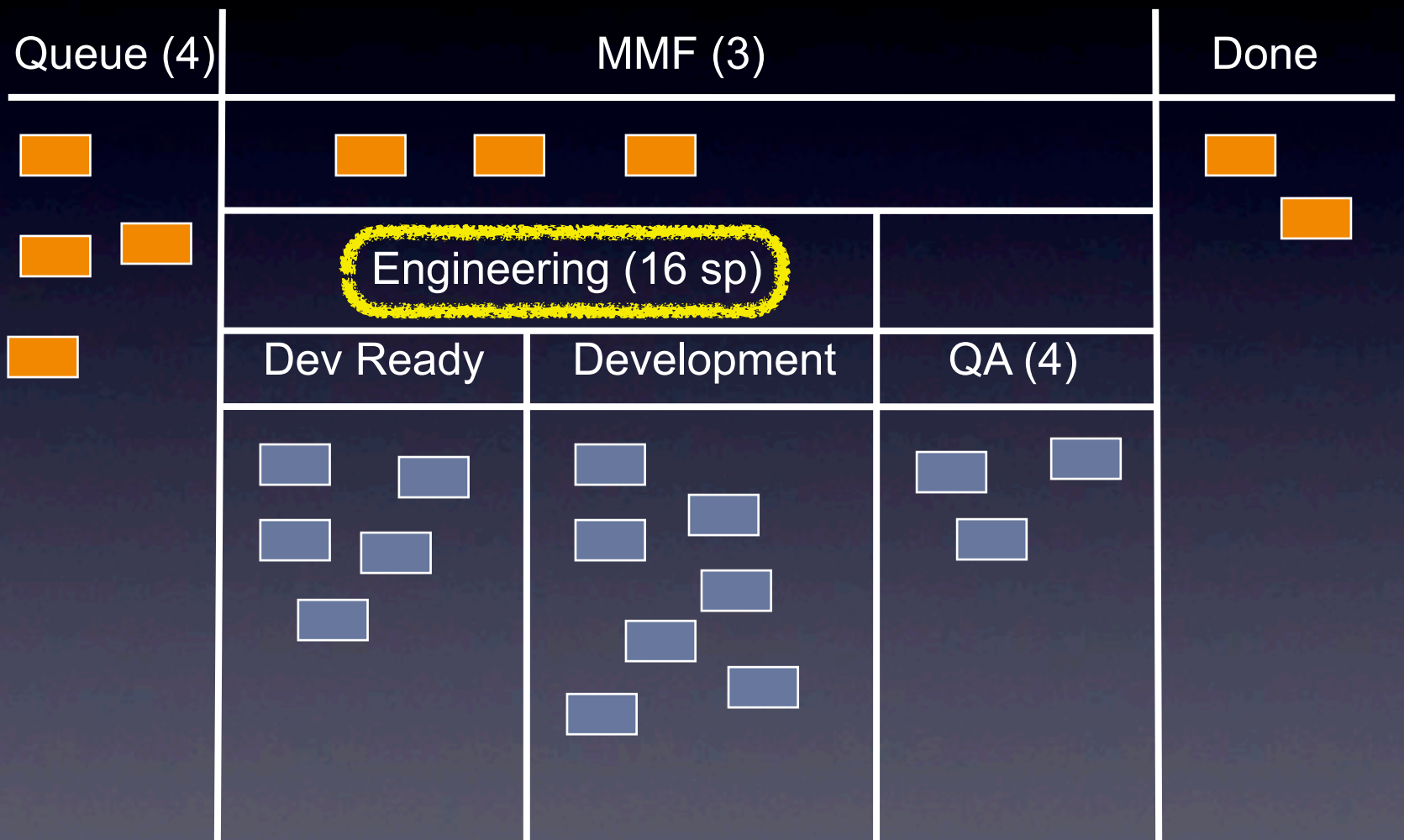
Handoffs and Specialisation

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.

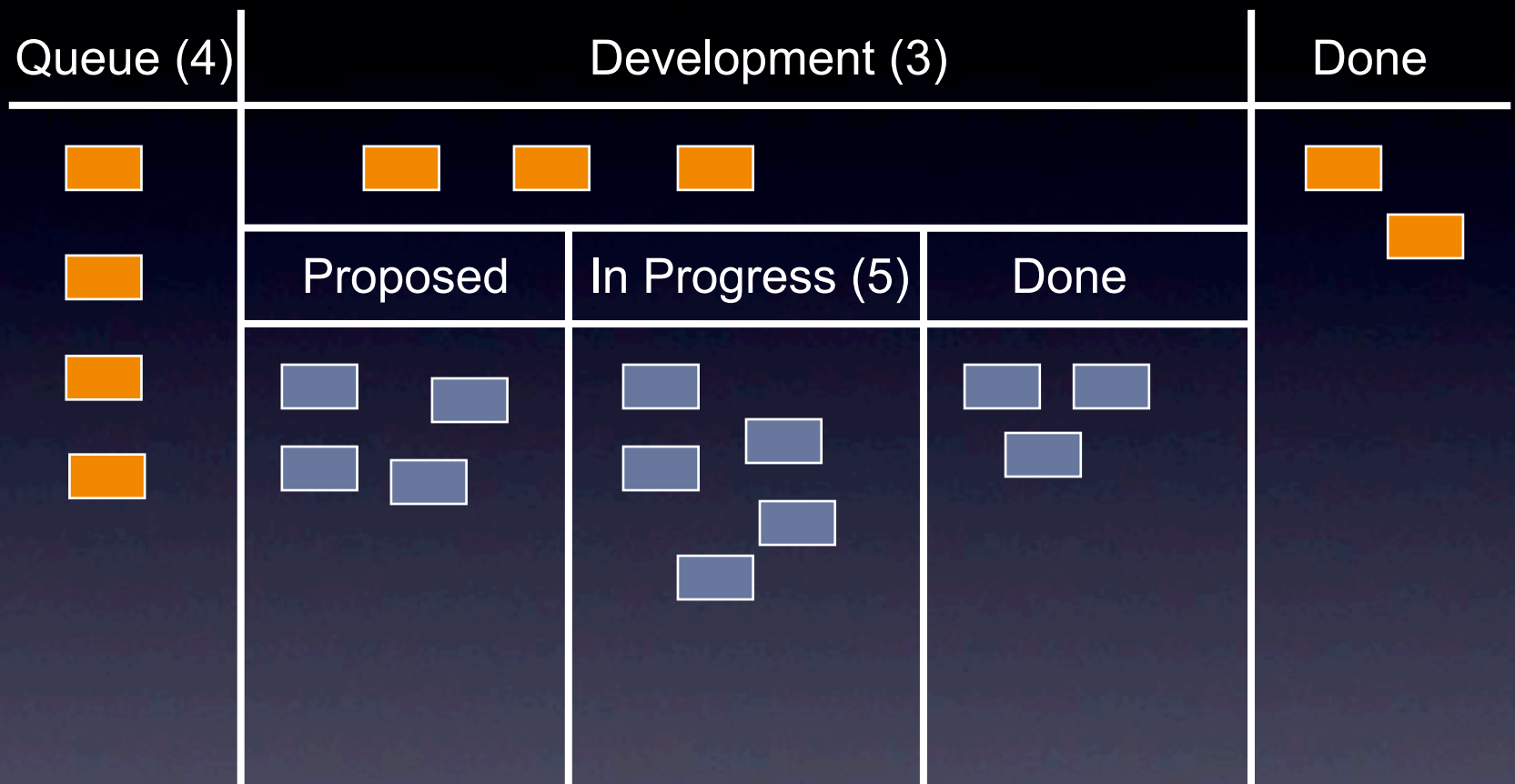
We can do that by defining some simple work standards or standard procedures for each state.

These can be drawn directly on the task board.

Limit By Story Points

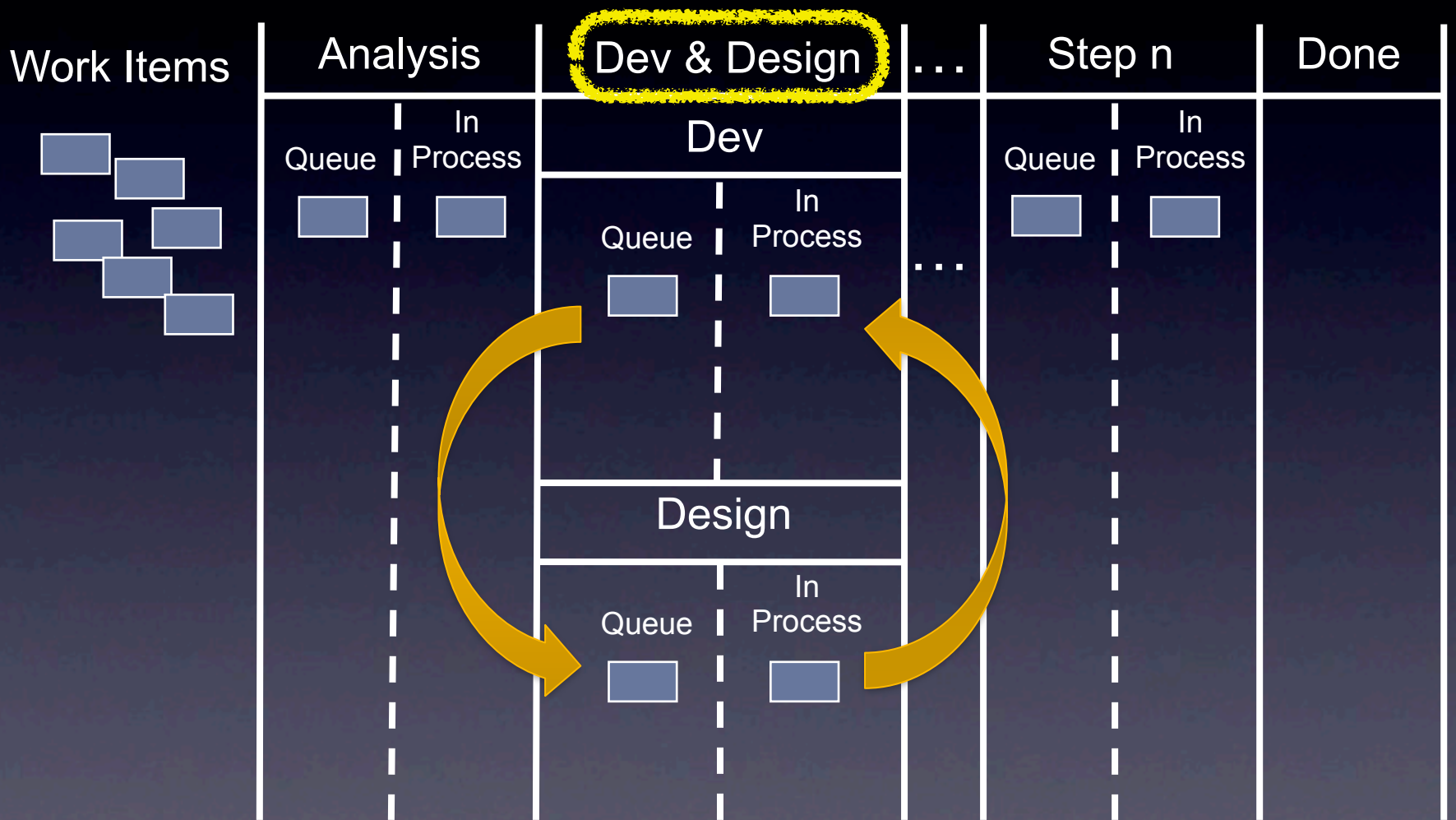


Two Tier Kanban



[3]

Non Sequential Flow



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.

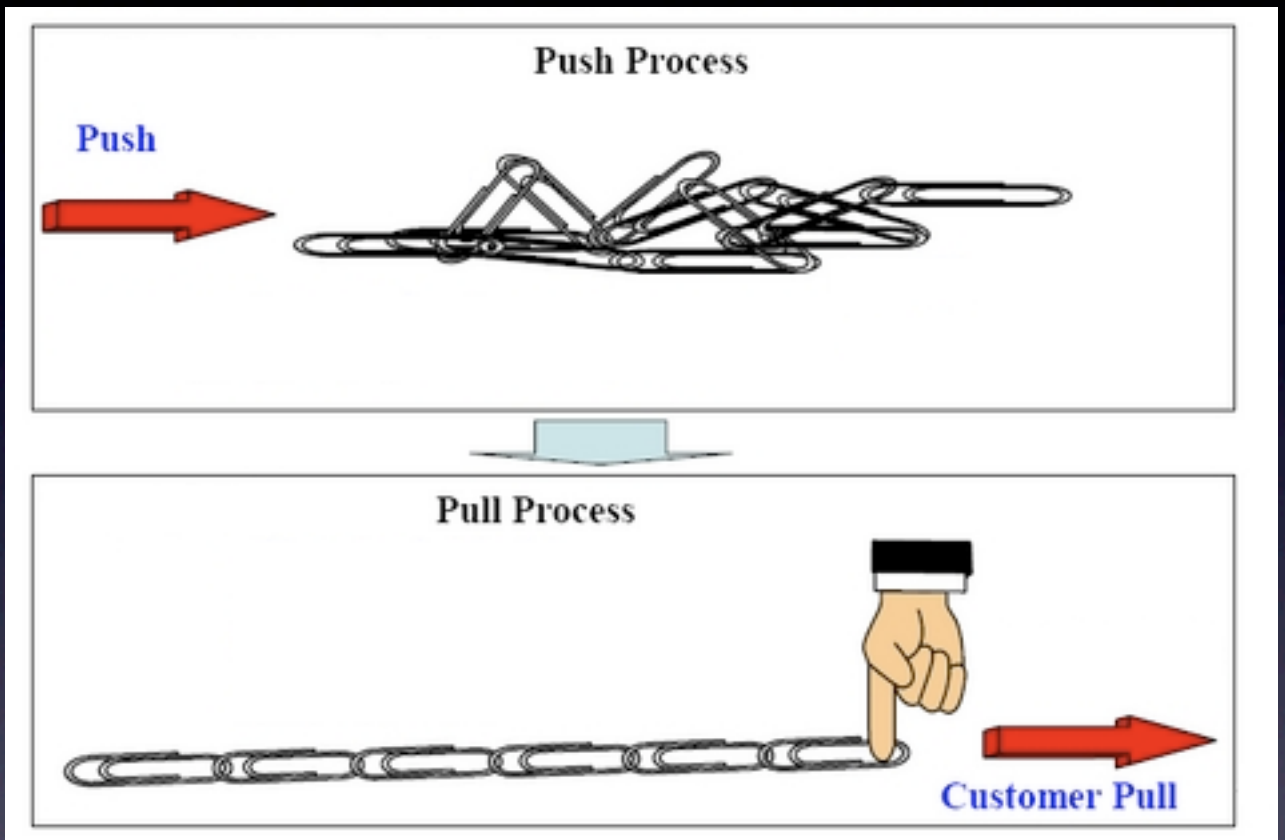
Personally, I don't think that there is a right or a wrong answer to this. You let the data make the decision for you. If releasing people to start something new improves the key indicators of throughput,
cycle time, and initial quality (defect rate), then why wouldn't you do it. Focusing solely on minimising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

[1]

Pull Work Not Push



Involving The Business

While Scrum can say they have a recourse for management pressure, there are two critical differences:

- Scrum treats the team's process as a black box to management
- The Scrum team's only real recourse is to abort the sprint (something that takes much more courage than would likely be available in an organisation where it actually needs to do it)

When the problem isn't the team's performance (as it often is not) then **including management** is the only way to improve the situation.

Planning

"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 Poppendiek

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

[2]

Quarterly Goals

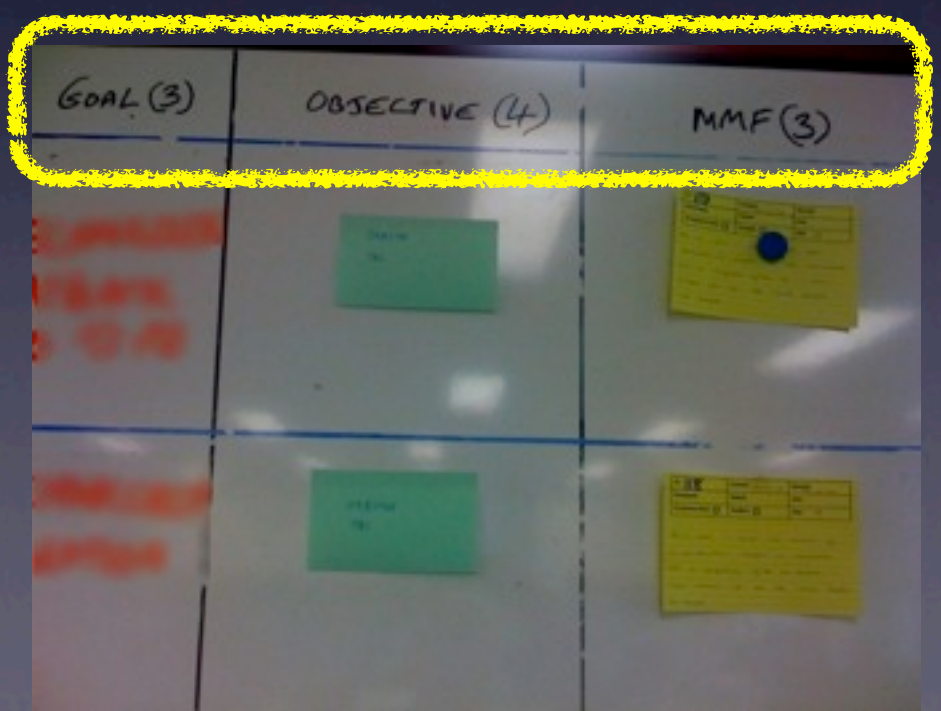
The Kanban board can be updated with two new states; Goals and Objectives.

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?



Targets

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*.

[2]

The Goal Comes First

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 Baggage 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, its 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 were 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 version control 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 is used.

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.

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)

[7]

Kanban vs Scrum Differences

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

[7]

Scrum

Scrum is more prescriptive than Kanban.

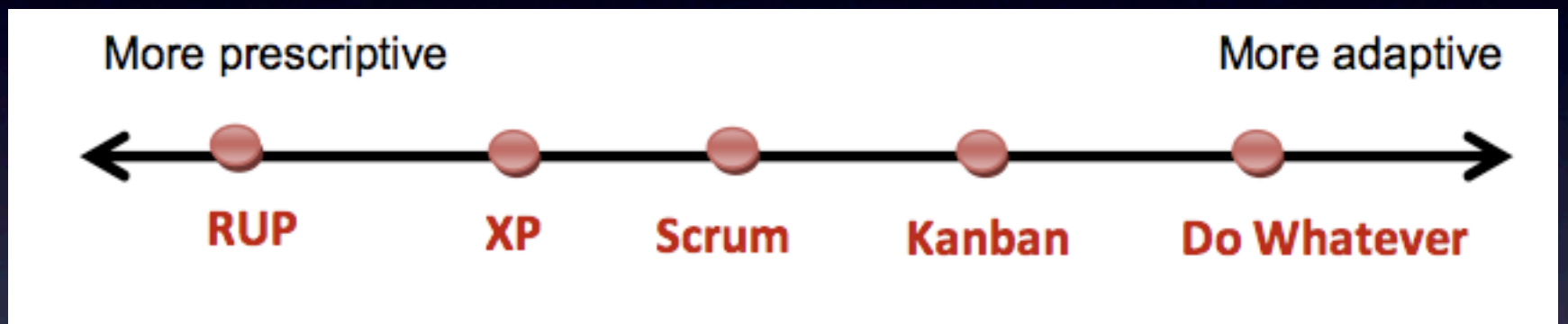
Agile methods are sometimes called lightweight methods, specifically because they are less prescriptive than traditional methods.

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.

Prescriptive vs Adaptive



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

Kanban Team Examples

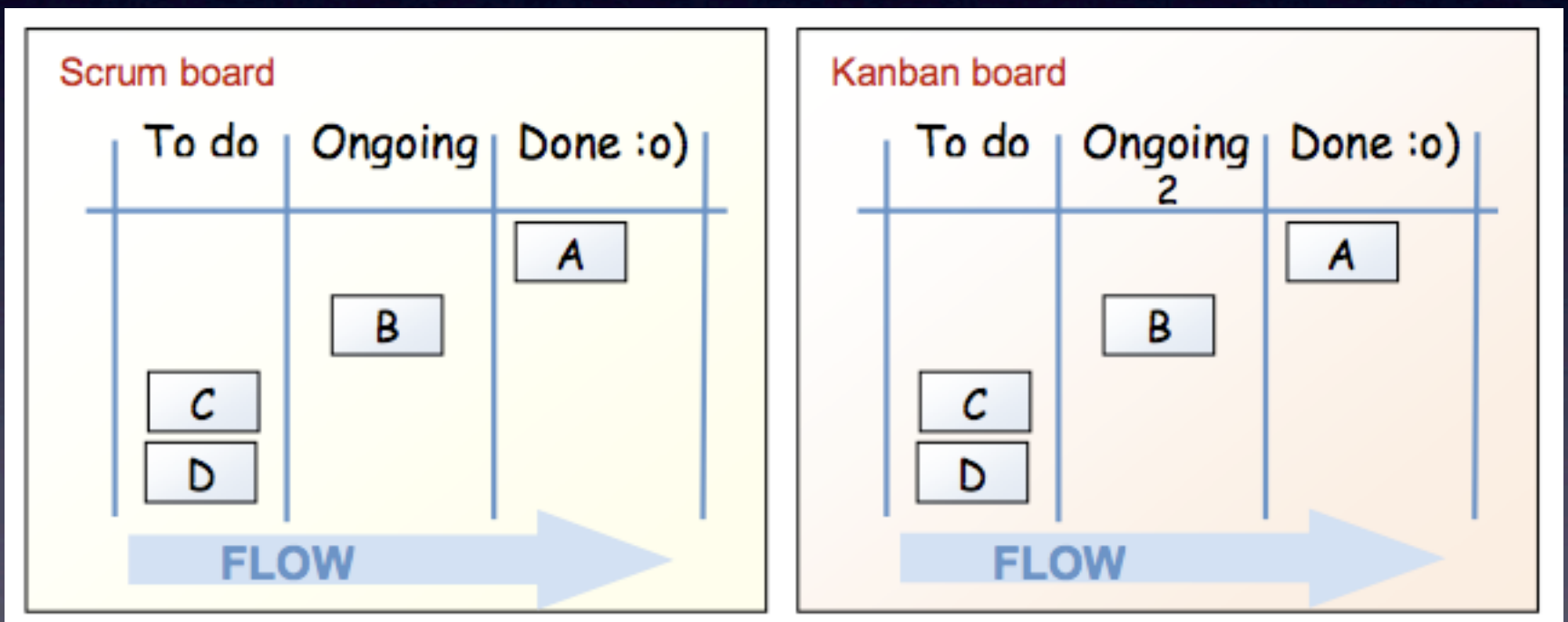
Kanban team #1 (single cadence): "We do Scrum iterations"

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."

WIP Limits

Kanban limits WIP per workflow state, Scrum limits WIP per iteration



Scrum Botherings

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.

Scrum Botherings

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/timeboxes.

As soon as an item is complete you pull in the next priority item from the queue. Therefore the wait time to inject a new request is minimal.

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.

Scrum Frustrations

Planning meeting attendance is poor as the customer doesn't have the time to spend 2 hours every 2 weeks planning.

You can 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 assumes you are starting a new project; sprint 0, calibration sprints, hardening sprints etc

Ongoing projects that are incrementally enhancing a product this doesn't apply.

You are able to bootstrap Kanban onto an existing project.

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 don't try to predict the future!

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.

[1]